XPath satisfiability in the presence of DTDs

XPath Satisfiability in the Presence of DTDs

MICHAEL BENEDIKT

Oxford University

WENFEI FAN

University of Edinburgh, Edinburgh, UK, and Bell Laboratories, Murray Hill, New Jersey

AND

FLORIS GEERTS

University of Edinburgh, Edinburgh, UK, and Hasselt University, and Transnational University of Limburg, Limburg, Belgium

Abstract. We study the satisfiability problem associated with XPath in the presence of DTDs. This is the problem of determining, given a query p in an XPath fragment and a DTD D, whether or not there exists an XML document T such that T conforms to D and the answer of p on T is nonempty. We consider a variety of XPath fragments widely used in practice, and investigate the impact of different XPath operators on the satisfiability analysis. We first study the problem for negation-free XPath fragments with and without upward axes, recursion and data-value joins, identifying which factors lead to tractability and which to NP-completeness. We then turn to fragments with negation but without data values, establishing lower and upper bounds in the absence and in the presence of upward modalities and recursion. We show that with negation the complexity ranges from PSPACE to EXPTIME. Moreover, when both data values and negation are in place, we find that the complexity ranges from NEXPTIME to undecidable. Furthermore, we give a finer analysis of the problem for particular classes of DTDs, exploring the impact of various DTD constructs, identifying tractable cases, as well as providing the complexity in the query size alone. Finally, we investigate the problem for XPath fragments with sibling axes, exploring the impact of horizontal modalities on the satisfiability analysis.

Categories and Subject Descriptors: F.4.3 [**Mathematical Logic and Formal Languages**]: Formal Languages—*Decision problems*; H.2.1 [**Database Management**]: Logical Design—*Data models*; I.7.2 [**Document and Text Processing**]: Document Preparation—*Markup languages*

General Terms: Algorithms, Languages, Theory

Additional Key Words and Phrases: DTDs, XML, XPath, Satisfiability, Containment

1. *Introduction*

XPath [Clark and DeRose 1999] has been widely used in XML query languages (e.g., XSLT, XQuery), specifications (e.g., XML Schema), update languages (e.g., Sur et al. [2004]), subscription systems (e.g., Chan et al. [2002]) and XML access control (e.g., Fan et al. [2004]). There is thus a need to study fundamental properties of the XPath language, and in particular to investigate static analyses of XPath queries.

The most basic static analysis of a query language is *satisfiability*: given a query in the language, does there exist a document (or database) on which it returns a nonempty answer? The satisfiability analysis of XPath is important for XML query and update optimization. Consider, for instance, an XML query construct commonly used: "*for* $\$x$ *in* p $c(\$x)$", where p is an XPath expression and $c(\$x)$ is a query or update. If one can decide, at compile time, that p is not satisfiable, then the unnecessary computation of $c(\$x)$ can be simply avoided. Furthermore, XPath satisfiability is critical as a foundation for other consistency problems, such as information leakage in security views [Fan et al. 2004], type-checking of transformations [Martens and Neven 2004], and consistency of XML specifications [Fan and Libkin 2002].

For relational languages, the satisfiability analysis is fairly trivial for *positive queries* such as conjunctive queries and positive fragments of Datalog, while it is trivially undecidable for the most prominent query languages with negation, such as relational calculus and stratified Datalog. For this reason, static analysis for relational languages has focused on the containment problem. In contrast, XPath satisfiability analysis is neither trivial nor futile. A variety of factors contribute to its complexity, such as the operators allowed in XPath queries, combinations of these operators, and their interaction with a schema.

1.1. The Analysis of XPath Satisfiability.

We now examine factors which XPath satisfiability analysis should take into account.

1.1.1. *XPath Fragments.*

The XPath 1.0 standard [Clark and DeRose 1999] offers a large array of operators. We consider several dichotomies, focusing on operators widely used in practice.

—**positive** vs. **nonpositive**: XPath allows qualifiers to be built up with a general negation operator, enabling it to express nonmonotone queries; queries without negation, referred to as *positive queries*, can be expressed in existential logic, while queries with negation may have existential and universal quantifiers;

—**downward** vs. **upward**: some XPath queries specify downward traversal (child, descendant), while others also have upward modalities (parent, ancestor);

—**recursive** vs. **nonrecursive**: some queries express navigation along the descendant (resp. ancestor) axis, while others only use the child (respectively, parent) axis;

—**qualified** vs. **nonqualified**: queries may or may not contain qualifiers (predicates testing properties defined in terms of other queries);

—**with** vs. **without data values**: queries may or may not contain comparisons of data values reached via different navigation paths, expressing data-value joins.

Even positive XPath queries defined in terms of simple operators may be unsatisfiable. Furthermore, from a practical point of view, many applications typically use only a limited set of operators. For example, XML Schema specifies integrity constraints with an XPath fragment that does not support upward modalities. This motivates the study of satisfiability for various XPath fragments (i.e., different combinations of operators). As will be seen later, with different combinations of these operators, the complexity of the satisfiability analysis ranges from PTIME to undecidable.

In this article, we study a class of XPath 1.0 queries, simply referred to as *XPath* in the sequel. This class supports the following operators: the wildcard (child) and the descendant-or-self-axis, the parent-axis and ancestor-or-self-axis, union, and qualifiers with data values and negation (not), as well as variants of sibling axes of XPath 1.0. We focus on these operators rather than new operators proposed by XPath 2.0 [W3C 2006] or XQuery [Chamberlin et al. 2001] since most queries found in practice are defined in terms of these operators. We also study various fragments of this class, supporting a subset of these operators.

1.1.2. *The Impact of Schema.* XML documents often come with a schema, typically a DTD. Any practical static analysis must take the schema into account. A DTD is far more complex than a relational schema, and it imposes structural constraints such as co-existence of certain siblings (by means of concatenation in the regular expressions within a DTD), exclusive relations on siblings (disjunction), as well as a limited form of negation (excluding certain children). These constraints interact with XPath queries in an intricate way. Indeed, as will be seen later, the satisfiability analysis is significantly impacted by the presence or absence of recursion (cycles) and disjunction in a DTD. The role of a schema gives the XPath satisfiability problem an additional dimension.

1.1.3. *XPath vs. Other Query Formalisms.* For relational query languages with negation, for example, SQL, the satisfiability problem is typically undecidable. For XPath the picture is far from clear, even in the presence of data values. While XPath is a rich language, it operates on documents whose underlying navigational structure is restricted to be a tree. Furthermore, XPath is a modal language, which, unlike relational calculus and the tree patterns of [Amer-Yahia et al. 2002], does not allow the explicit use of free variables. As a result XPath queries can "see" only one

node at a time when navigating a tree. This restriction becomes more prominent in the presence of data values or negation, since it allows only a restricted form of data joins to be expressed. This makes the expressive power of XPath quite distinct from other XML query languages, particularly from XML query algebras [Jagadish et al. 2002; Paparizos et al. 2004]. The limited expressiveness significantly decreases the complexity of the satisfiability analysis in the presence of negation. We will show that in the absence of data values the satisfiability problem for XPath fragments with negation is in EXPTIME, and if recursive axes are further disallowed, it is in PSPACE. This contrasts with first-order logic over trees, the most natural example of a tree query language with explicit variables: there the satisfiability problem has nonelementary complexity [Stockmeyer 1974], even when only the child relation is present.

Taken together, these factors lead to a rich spectrum of languages and satisfiability problems with features quite distinct from the containment analysis and relational satisfiability.

1.2. MAIN RESULTS. Here we present a comprehensive picture of the satisfiability problem for a variety of XPath fragments in the presence and in the absence of DTDs.

1.2.1. *Positive XPath.* We begin with a minimal XPath fragment with only the child axis in the presence of DTDs. We then investigate the impact of adding qualifiers, union, upward traversal (the parent axis), recursion (the descendant and ancestor axes) and data values, one at a time, establishing the complexity of the satisfiability problem for each of these fragments. We show that the complexity ranges from PTIME to NP-complete (Section 4).

1.2.2. *XPath with Negation.* We then investigate a minimal XPath fragment with negation. Since negation makes sense only in the presence of qualifiers, we begin with the XPath fragment with qualifiers, negation and the child axis. We show that the satisfiability problem for this fragment is PSPACE-complete in the presence of DTDs. We then look at the impact of adding upward modality and recursion, one at a time. The complexity here will vary between PSPACE-complete and EXPTIME-complete. Finally, we show that the combination of data values and negation makes a big difference, by adding data values to these fragments. We find that the complexity is in NEXPTIME in the absence of recursive and upward axes, and is undecidable in the general case (Section 5).

1.2.3. *Particular DTDs.* To explore the impact of different DTD constructs on XPath satisfiability analysis and to understand the interaction between XPath queries and DTD constructs, we also investigate XPath satisfiability under various restricted DTDs. More specifically, we consider nonrecursive DTDs, fixed DTDs, and disjunction-free DTDs (i.e., the regular expressions in a DTD do not contain disjunction; we do not consider other kinds of restrictions like star-free, 1-unambiguous or order-independent DTDs). We show that the worst-case complexity does not diminish when the DTD is fixed, but can decrease dramatically in the absence of DTD recursion or disjunction. We also revisit the satisfiability problem for all these fragments *in the absence of DTDs*. We show that for positive XPath, the absence of DTDs simplifies the satisfiability analysis, but this is not the case for those fragments with negation (Section 6).

1.2.4. *Reductions between Problems.* We also provide basic results for the connections between XPath satisfiability and XPath containment, between XPath satisfiability analysis in the presence of DTDs and that in the absence of DTDs, and between XPath satisfiability under arbitrary DTDs and that under a simple normal form of DTDs. These results allow us to restrict to special cases of the satisfiability problem in proving results about both satisfiability and containment (Section 3).

1.2.5. *XPath with Sibling Axes.* Finally, we revisit the satisfiability problem for XPath fragments that support sideways traversal, namely, the (immediate) following-sibling and preceding-sibling axes. The objective is to explore the impact of the sibling axes on the satisfiability analysis, compared to their vertical counterparts. We establish complexity results for a variety of XPath fragments with the sibling axes, in the presence of DTDs, under restricted DTDs, and in the absence of DTDs (Section 7).

We establish matching upper and lower bounds in all the cases without data values and sibling axes, and several complexity results for fragments with data values or sibling axes. To our knowledge, this work is the first detailed theoretical study of XPath satisfiability in the presence of DTDs, under restricted DTDs, and in the absence of DTDs. A variety of techniques are used to prove these results, including two-way alternating automata, rewriting systems, finite-model theoretic constructions, bounded-branching results, and a wide range of reductions.

1.3. RELATED WORK. Static analysis of XML queries has for the most part been developed along the lines laid down in the relational theory. In the relational setting, the emphasis has been on the containment problem (given Q_1 and Q_2, does Q_1 always return a subset of Q_2?) for positive queries. There is limited practical or theoretical motivation for the satisfiability study in the relational case: a user can never propose a (union of) conjunctive query that is "nonsensical", so there is no pressing need to check this; and for more general relational queries, a complete satisfiability checking is theoretically impossible. In analogy with this, prior work on XPath static analysis has mostly concentrated on the containment problem for *positive* XPath [Deutsch and Tannen 2005; Miklau and Suciu 2004; Neven and Schwentick 2006; Wood 2002; Marx 2004]. These (except Neven and Schwentick [2006] and Marx [2004]) are the analogs of positive queries for tree-like data, asserting the existence of a certain kind of trees as a substructure of the document. While the satisfiability problem is subsumed by the complement of the containment problem for XPath, we shall see that the upper bounds on satisfiability derived from previous work on containment for positive XPath fragments are far from tight. Similarly, we shall see that our bounds do not imply anything about the containment analysis for those *positive* fragments studied previously.

While Neven and Schwentick [2006] proves bounds on containment in the presence of negation and data values (without upward axes), it does not consider the general XPath negation operator, and instead negation is tied to particular equality comparisons. Data values in Neven and Schwentick [2006] are introduced in the form of variables. Variables are not an XPath 1.0 notion, and they change the modal nature of the language dramatically. Compared to XPath 1.0, the expressiveness and succinctness of these variables and data values are not clear. Since this semantics of negation and data values is different from ours, our results do not imply the results of Neven and Schwentick [2006], and vice versa.

Marx [2004] is concerned principally with extensions of XPath, but contains bounds on equivalence and satisfiability for the largest fragment we consider that *lacks* data value equality. Marx [2004] proves an EXPTIME upper bound on containment for an extension of XPath, which implies an EXPTIME bound on satisfiability for the fragment that supports the child, wildcard, descendant-or-self, parent, ancestor-or-self axes as well as union, negation and qualifiers. Indeed, the results of Marx [2004] imply an EXPTIME upper bound on the extension of this fragment with the sibling axes in the presence of specialized DTDs (roughly speaking, XML Schema), rather than just DTDs. A corresponding EXPTIME hardness result can be derived from a lower bound on query equivalence of the "XCore" language of Marx [2004] (we shall elaborate on this in Section 5).

For fragments with the XPath negation operator, upward modalities and data-value joins, the containment problem has not been studied, with or without DTDs. We shall show (Proposition 3.2) that in the presence of *negation*, our upper and lower bounds for the satisfiability problem can be carried over to the containment problem for the corresponding fragments, and thus establish the first results for the containment problem for those fragments with negation, upward modalities and data-value joins.

As we have remarked, XPath satisfiability is both theoretically interesting and practically important, in contrast to its relational counterpart. However, to our knowledge, the satisfiability problem has only been studied for (positive) tree-patterns [Lakshmanan et al. 2004] in the presence of (restricted) DTDs, and for certain XPath fragments in the absence of DTDs [Hidders 2004] and (as mentioned above) in Marx [2004] where EXPTIME upper and lower bounds are proved on XPath with negation and all axes, but without data values, in the presence of DTDs. The satisfiability problem for a number of practical and interesting fragments, especially those with negation and data values, has not been studied, with or without DTDs.

Hidders [2004] differs from our work in both the set of operators it considers (e.g., without data values), and in that it assumes the absence of DTDs. It gives PTIME bounds in the presence of qualifiers, sibling axes, upward axes, and a root test. Our results suffice to show that these bounds do not hold in the presence of DTDs. The proofs of Hidders [2004] also imply that the satisfiability problem is NP-complete when downward axes are supplemented by an intersection operator. The intersection operator is not available in XPath 1.0, so we do not consider it here. Finally, Hidders [2004] shows that satisfiability is NP-hard in the presence of a complement operator, which is again not supported by XPath. Instead, we consider here the XPath negation operator, proving both lower and upper bounds. Note that our results would also carry over to show that XPath fragments with all of the features of Hidders [2004] is in EXPTIME, in the presence of DTDs.

Lakshmanan et al. [2004] considers a tree pattern formalism with expressiveness incomparable to XPath. This expresses tree-shaped, positive queries, with data value equality and inequality along with a node-equality test. Note that node-equality can be used to simulate the intersection operation of Hidders [2004]. Lakshmanan et al. [2004] shows that the satisfiability problem is NP-complete for several restrictions of this pattern language in the absence of DTDs. It also investigates the satisfiability of tree pattern queries with limited use of data joins (these can only occur "conjunctively") and node equality and inequality under non-recursive disjunction-free DTDs. Since these results impose severe syntactic restrictions, all of which make

sense only within the particular pattern formalism rather than in XPath, and it is difficult to compare the results with ours on positive XPath. Lakshmanan et al. [2004] does not deal with negation, nor can the XPath negation operator be simulated in the formalism of Lakshmanan et al. [2004].

Minimization, rewriting and optimization are studied for tree patterns and XPath [Amer-Yahia et al. 2002; Gottlob et al. 2005; Olteanu et al. 2002; Wood 2001]. Expressiveness of XPath is investigated in Benedikt et al. [2005], Milo et al. [2003], Murata [2001], and Neven and Schwentick [2000]. No bounds for satisfiability follow from these works.

There are several powerful logical formalisms on trees for which satisfiability is decidable, principally Monadic Second Order Logic (MSO) [Thatcher and Wright 1968]. All the XPath fragments we consider that omit data-value equality can be translated into MSO, thus giving a decision procedure. However, MSO on trees (and even first-order logic) has a non-elementary satisfiability problem [Stockmeyer 1974]. For the fragments with data values we know of no semantics-preserving translation into an existing formalism.

The bounds of Marx [2004] are closely-related to work on the satisfiability problem for Propositional Dynamic Logic (PDL). Like XPath, PDL allows for the definition of binary relations (in PDL terms, programs) as well as unary relations, and like XPath PDL is a modal language. Satisfiability for deterministic PDL with converse is known to be EXPTIME-complete [Vardi and Wolper 1986]. Although PDL is defined on Kripke structures, rather than trees, proofs of the satisfiability bounds for deterministic PDL with converse can be modified to give bounds on satisfiability of the same language on binary trees; the satisfiability of the XPath fragment with negation, all axes but without data value joins can be reduced to this problem [Marx 2004]. Afanasiev et al. [2005] discusses a variant of PDL directly applicable to ordered trees, which is strictly more expressive than the largest fragment without data joins we consider. The complexity bounds in Afanasiev et al. [2005] for this variant of PDL imply (as does Marx [2004]) EXPTIME-completeness for this particular fragment. For the other main fragments of XPath that we study here, we know of no correspondence with a fragment or extension of PDL.

1.4. ORGANIZATION. Section 2 reviews DTDs and XPath fragments. Sections 3, 4, 5, 6 and 7 establish technical results as outlined above. Section 8 summarizes our main results.

2. *Notations: DTDs and XPath Fragments*

In this section, we first review Document Type Definitions (DTDs [Bray et al. 1998]) and then define the fragments of XPath [Clark and DeRose 1999] studied in this article.

2.1. DTDs. Without loss of generality, we represent a DTD D as (Ele, Att, P, R, r), where (1) Ele is a finite set of *element types*, ranged over by $A, B, \ldots$; (2) r is a distinguished type in Ele, called the *root type*; (3) P is a function that defines the element types: for each A in Ele, $P(A)$ is a regular expression over Ele; we refer to $A \rightarrow P(A)$ as the *production* of A; (4) Att is a finite set of *attribute names*, ranged over by $a, b, \ldots$; and (5) R defines the attributes: for each A in Ele,

$R(A)$ is a subset of Att. We do not consider additional DTD features such as default values and attribute domains.

An XML document is typically modeled as a (finite) node-labeled ordered tree [Bray et al. 1998], with nodes additionally annotated with values for attributes. We refer to this as an *XML tree*. An XML tree T *satisfies* (or *conforms to*) a DTD $D = (Ele,\ Att,\ P,\ R,\ r)$, denoted by $T \models D$, if (1) the root of T is labeled with r; (2) each node n in T is labeled with an Ele type A, called an A *element*; the label of n is denoted by $\mathsf{lab}(n)$; (3) each A element has a list of *children* (*subelements*) such that their labels are in the regular language defined by $P(A)$; and (4) for each $a \in R(A)$, each A element n has a unique *a attribute value* which we denote by $n.a$. We call T an *XML tree* of D if $T \models D$.

We also study the following special forms of DTDs.

A *normalized DTD* is a DTD in which for each A in Ele, $P(A)$ is of the following form:

$$\alpha ::= \varepsilon \quad | \quad B_1, \ldots, B_n \quad | \quad B_1 + \cdots + B_n \quad | \quad B^*$$

where ε is the empty word, B_i is a type in Ele (referred to as a *child type* of A), and '$+$', '$,$' and '$*$' denote *disjunction*, *concatenation* and the *Kleene star*, respectively (here we use '$+$' instead of '$|$' to avoid confusion). We will see later (Proposition 3.3) that there is often no loss of generality in restricting to normalized DTDs.

A DTD D is said to be *disjunction-free* if for any element type $A \in Ele$, $P(A)$ does not contain disjunction '$+$'.

A DTD D is *recursive* if the dependency graph of D (which contains an edge (A, B) iff B is in $P(A)$) has a cycle.

A recursive DTD D may not have any XML tree T such that $T \models D$. This is because some element type A in D is *non-terminating*, i.e., there exists no finite subtree rooted at A that satisfies D. One can determine whether an element type A in D is terminating or not in $O(|D|)$ time, where $|D|$ is the size of D. Indeed, this problem can be reduced to the emptiness problem for context-free grammars, which can be determined in linear time (cf. Hopcroft and Ullman [2000]). Thus to simplify the discussion, in the sequel we assume that all element types in a DTD are terminating. All the complexity results (lower bounds and upper bounds) in this article remain unchanged in the presence of nonterminating element types.

Example 2.1. Consider an instance $\phi = C_1 \wedge \cdots \wedge C_n$ of 3SAT (cf. Papadimitriou [1994]), where $C_i = l_{(i,1)} \vee l_{(i,2)} \vee l_{(i,3)}$, $l_{i,j}$ is a literal of the form x_s or $\bar{x}_s$, and x_s is a propositional variable. Assume that the variables in ϕ are $x_1, \ldots, x_k$. Given ϕ, we define a DTD $D_\phi = (Ele, Att, P, R, r)$, where $Ele = \{r, T, F, X_1, \ldots, X_k\}$, $Att = \emptyset$, and

$$P:\ r \to X_1, \ldots, X_k, \qquad X_i \to T + F, \text{for } i \in [1, k], \qquad T \to \varepsilon, \qquad F \to \varepsilon;$$
$$R:\ R(A) = \emptyset, \quad \text{for any } A \in Ele$$

An XML tree of D_ϕ lists all the variables X_i under the root, and gives a truth value (T or F) under each X_i. The DTD is normalized and nonrecursive; it is not disjunction free.

2.2. XPath Fragments. Over an XML tree, an XPath query specifies the selection of nodes in the tree. Assume a (possibly infinite) alphabet Σ of labels. We define the largest class of XPath queries considered in this article, referred to

as $\mathcal{X}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \cup, [\ ], =, \neg)$, syntactically as follows:

$$p \ ::= \ \varepsilon \ | \ l \ | \ \downarrow \ | \ \downarrow^* \ | \ \uparrow \ | \ \uparrow^* \ | \ p/p \ | \ p \cup p \ | \ p[q],$$

where ε and l denote the empty path (the *self-axis*) and a label in Σ (the *child-axis*); '$\downarrow$' and '$\downarrow^*$' stand for the wildcard (*child*) and the *descendant-or-self-axis*, while $\uparrow$ and $\uparrow^*$ denote the *parent-axis* and *ancestor-or-self-axis*, respectively; '/' and '$\cup$' denote concatenation and union, respectively; and finally, q in $p[q]$ is called a *qualifier* and is defined by:

$$q ::= \ p \mid \mathsf{lab} = A \mid p/@a \ \mathsf{op} \ \text{`}c\text{'} \mid p/@a \ \mathsf{op} \ p\text{'}/@b \mid \ q_1 \wedge q_2 \mid q_1 \vee q_2 \mid \neg q,$$

where p, p' are as defined above, A is a label in Σ, op is either '$=$' or '$\neq$', a, b stand for attributes, c is a constant (string value), and $\wedge$, $\vee$, $\neg$ stand for and (conjunction), or (disjunction) and not (negation), respectively.

The significant deviation from the XPath 1.0 standard is that the latter restricts the union operator to occur only at top-level; in order to have a more compositional language we drop this restriction, as do the prior studies of XPath (e.g., Marx and de Rijke [2005] and Benedikt et al. [2005]). The expressiveness of the fragments we study is not affected by this. Perhaps more surprisingly, neither are the complexity bounds: upper bounds for our language clearly imply the corresponding bounds for the more restricted one that forbids nested union, while all of our lower bounds can be seen to use only top-level union.

A query p in $\mathcal{X}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \cup, [\ ], =, \neg)$ over an XML tree T is interpreted as a binary predicate on the nodes of T, while a qualifier is interpreted as a unary predicate. More specifically, for any nodes n in T, T *satisfies* p *at* n if and only if $T \models \exists n' \ p(n, n')$, where $T \models p(n, n')$ and the associated version for qualifiers, $T \models q(n)$, are defined inductively on the structure of p, q, as follows.

—if $p = \varepsilon$, then $n = n'$;
—if $p = l$, then n' is a child of n, and is labeled l;
—if $p = \downarrow$, then n' is a child of n, regardless of its label;
—if $p = \downarrow^*$, then n' is either n or a descendant of n;
—if $p = \uparrow$, then n' is the parent of n;
—if $p = \uparrow^*$, then n' is either n or an ancestor of n;
—if $p = p_1/p_2$, then there exists a node v in T such that $T \models p_1(n, v)$ and $T \models p_2(v, n')$;
—if $p = p_1 \cup p_2$, then $T \models p_1(n, n')$ or $T \models p_2(n, n')$;
—if $p = p_1[q]$, then $T \models p_1(n, n')$ and $T \models q(n')$, where q is a unary predicate of the following cases:
  —if q is p_2, then there exists a node n'' in T such that $T \models p_2(n', n'')$;
  —if q is $lab() = A$, then the label of n' is A;
  —if q is $p_2/@a \ \mathsf{op} \ \text{`}c\text{'}$, then there exists a node $n_1$ in $T$ such that $T \models p_2(n', n_1)$, $n_1$ has attribute $a$ and $n_1.a \ \mathsf{op} \ \text{`}c\text{'}$, where $n_1.a$ denotes the value of the a attribute of n_1;
  —if q is $p_2/@a \ \mathsf{op} \ p_2'/@b$, then there exist two nodes n_1 and n_2 in T such that $T \models p_1(n', n_1)$, $T \models p_2(n', n_2)$, n_1 (resp. n_2) has attribute a (resp. b), and $n_1.a \ \mathsf{op} \ n_2.b$;
  —if q is $q_1 \wedge q_2$, then $T \models q_1(n')$ and $T \models q_2(n')$;
  —if q is $q_1 \vee q_2$, then $T \models q_1(n')$ or $T \models q_2(n')$;

—if q is $\neg q'$, then $T \not\models q'(n')$; for instance, if q is $\neg p_2$, then there does not exist a node n'' in T such that $T \models p_2(n', n'')$, i.e., $T \models \forall n'' \neg p_2(n', n'')$.

Here n is referred to as the *context node*. If $T \models p(n, n')$, then we say that n' is *reachable* from n via p. We use $n[\![p]\!]$ to denote the set of all the nodes reached from n via p, that is, $n[\![p]\!] = \{n' \mid n' \in T,\ T \models p(n, n')\}$.

We say that an XML tree T *satisfies* a query p, denoted by $T \models p$, if and only if $T \models \exists n\ p(r, n)$, where r is the root of T. In other words, $r[\![p]\!] \neq \emptyset$, i.e., the set of nodes reachable from the root of T via p is nonempty. Similarly, we talk about T satisfying a qualifier q if $T \models q(r)$. To simplify the discussion, we focus on the satisfiability of XPath queries applied to the root of T (note that its corresponding containment problem relates to the root equivalence [Benedikt et al. 2005]). It should be mentioned that the complexity results of this paper remain intact for arbitrary context nodes.

We also investigate various fragments of the language $\mathcal{X}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \cup, [\,], =, \neg)$. We denote a fragment $\mathcal{X}$ by listing the operators supported by $\mathcal{X}$: the presence or absence of negation '$\neg$', data values '$=$' (indicating op, including both '$=$' and '$\neq$'), upward traversal '$\uparrow$' ('$\uparrow^*$'), recursive axis '$\downarrow^*$' ('$\uparrow^*$'), qualifiers '[ ]', wildcard '$\downarrow$', and union and disjunction '$\cup$' (the absence of '$\cup$' indicates that *neither* union '$\cup$' *nor* disjunction '$\vee$' is allowed). The concatenation operator '$/$' *is included in all the fragments by default*.

For example, a small fragment with negation is $\mathcal{X}(\downarrow, [\,], \neg)$ (note that '$\neg$' can only appear in qualifiers), and the largest positive fragment is $\mathcal{X}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \cup, [\,], =)$ (it should be mentioned that limited negation defined by '$\neq$' is allowed in this fragment, while its subclass without data values, namely, $\mathcal{X}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \cup, [\,])$, allows neither '$=$' nor '$\neq$').

All these fragments have been found useful in practice. For example, $\mathcal{X}(\downarrow, \downarrow^*, \cup)$ is used by XML Schema to specify integrity constraints, and $\mathcal{X}(\downarrow, \downarrow^*, [\,])$ is the class of tree-pattern queries studied in Amer-Yahia et al. [2002], Olteanu et al. [2002], and Wood [2001].

Example 2.2. Recall the 3SAT instance ϕ and the DTD D_ϕ given in Example 2.1. One can use a query p in $\mathcal{X}(\cup, [\,])$ to encode ϕ, where p is specified as:

$$p = \varepsilon[q_1 \wedge \cdots \wedge q_n], \quad q_i = \mathsf{XP}(l_{i,1}) \vee \mathsf{XP}(l_{i,2}) \vee \mathsf{XP}(l_{i,3}),$$
$$\text{where } \mathsf{XP}(l_{j,j}) = X_s/T \text{ if } l_{i,j} \text{ is } x_s, \text{ and } \mathsf{XP}(l_{j,j}) = X_s/F \text{ if } l_{i,j} \text{ is } \bar{x}_s.$$

Indeed, ϕ is satisfiable iff there is an XML tree T of the DTD D_ϕ such that $T \models p$. □

Example 2.3. Consider the DTD $D = (Ele, Att, P, R, r)$, where $Ele = \{r, A\}$, $Att = \emptyset$, $P : r \to A^*$, and R: $R(r) = R(A) = \emptyset$. Consider the XPath query $p = B$. Then it is clear that there exists no XML tree T of the DTD D such that $T \models p$. □

3. *The Satisfiability Problem*

We are interested in the satisfiability problem for XPath queries considered together with a DTD: that is, whether a given XPath query p and a DTD D aresatisfiable by

an XML tree. We say that an XML tree T *satisfies* p *and* D, denoted by $T \models (p, D)$, if $T \models p$ and $T \models D$. If such a T exists, we say that (p, D) is *satisfiable*.

For a fragment $\mathcal{X}$ of XPath, the *XPath satisfiability problem* $\mathsf{SAT}(\mathcal{X})$ is stated as follows:

PROBLEM:	$\mathsf{SAT}(\mathcal{X})$
INPUT:	A DTD D, an XPath query p in $\mathcal{X}$.
QUESTION:	Is there an XML document T such that $T \models (p, D)$?

Below we present several basic results for $\mathsf{SAT}(\mathcal{X})$.

3.1. SATISFIABILITY BASICS. The *satisfiability problem for a fragment* $\mathcal{X}$ *in the absence of DTDs* is the problem of determining, given any query p in $\mathcal{X}$, whether there is an XML tree T such that $T \models p$.

This version of the satisfiability problem for $\mathcal{X}$ is actually a special case of $\mathsf{SAT}(\mathcal{X})$, since it can be reduced to $\mathsf{SAT}(\mathcal{X})$ when the input DTD is fixed to range over DTDs of the form $D_p = (Ele_p, Att_p, P_p, R_p, r_p)$, where (1) Ele_p consists of a distinct label X as well as all the labels A mentioned in p in the form of a subquery A or a qualifier $\mathsf{lab}() = A$; (2) Att_p consists of all the attribute names a, b mentioned in p in the form of a qualifier $p/@a\ \mathsf{op}\ `c'$ or $p/@a\ \mathsf{op}\ p'/@b$; (3) for each $A \in Ele_p$, the production for A is defined to be $A \to (A_1 + \cdots + A_n)^*$, where $Ele_p = \{A_1, \ldots, A_n\}$; (4) $R_p(A)$ is defined to be Att_p; and (5) r_p is one of the A_i's in Ele_p. The connection between this satisfiability problem and $\mathsf{SAT}(\mathcal{X})$ is encapsulated in the following.

PROPOSITION 3.1. *For any fragment* $\mathcal{X}$ *of XPath queries defined above and any query* p *in* $\mathcal{X}$*, there exists an XML tree* T *such that* $T \models p$ *if and only if there exists an XML tree* T' *such that* $T' \models (p, D)$*, where* D *has the form of* D_p *given above.*

PROOF. If there exists an XML tree T' such that $T' \models (p, D)$, then obviously $T' \models p$. Conversely, suppose that there exists an XML tree T such that $T \models p$. Observe that all the labels in Ele_p appear in p except the distinct label X, and similarly, that all the attribute names in Att_p appear in p. Define an XML tree T' by re-labeling element nodes and removing attributes in T as follows: (1) For each element n in T, if $\mathsf{lab}(n)$ is not in Ele_p, then rename the label of n to be X in T'; (2) For each attribute of n, if its name is not in Att_p, then remove the attribute. Then, one can easily verify that $T' \models D$ by the definition of D and the construction of T'. Furthermore, $T' \models p$ since the renaming and deletions preserve the truth values of subqueries and qualifiers at each node n in T. Indeed, for any node n in T, and for any subquery or qualifier p' of the form l, $\mathsf{lab}() = A$, $p_1/@a\ \mathsf{op}\ `c'$ or $p_1/@a\ \mathsf{op}\ p_2/@b$ (or their negation), T satisfies p' at n if and only if T' satisfies p' at n. Based on this, one can show that $T \models p$ if and only if $T \models p'$, by a straightforward induction on the structure of p. □

For a query p, there are at most $O(|p|)$ many such DTDs D (by allowing r_p to range over all the element types in Ele_p), and the size of such D is in $O(|p|^2)$. As a result, since all the upper bounds for $\mathsf{SAT}(\mathcal{X})$ established in this paper are with respect to complexity classes containing PTIME, they also hold for the satisfiability problem for $\mathcal{X}$ in the absence of DTDs. However, we shall see that for some

fragments $\mathcal{X}$, the complexity for its satisfiability problem in the absence of DTDs can be much lower than its counterpart for the same fragment $\mathcal{X}$ in the presence of DTDs.

In the sequel, we also refer to the satisfiability problem for $\mathcal{X}$ in the absence of DTDs as $\mathsf{SAT}(\mathcal{X})$ when it is clear from the context.

The *containment problem* for a fragment $\mathcal{X}$ in the presence of DTDs, denoted by $\mathsf{CNT}(\mathcal{X})$, is the problem to determine, given any queries $p_1, p_2 \in \mathcal{X}$ and a DTD D, whether or not for any XML tree T of D, $r[\![p_1]\!] \subseteq r[\![p_2]\!]$, where r is the root of T. That is, whether or not the answer to p_1 is contained in the answer to p_2 over all the XML trees of D. If this holds, then we say that $p_1 \subseteq p_2$ *under* D.

For any fragment $\mathcal{X}$, $\mathsf{SAT}(X)$ is reducible to the complement of $\mathsf{CNT}(\mathcal{X})$. Indeed, for any query $p \in X$ and DTD D, (p, D) is satisfiable if and only if $p_1 \not\subseteq \emptyset_D$ under D, where $\emptyset_D$ is a special query that returns an empty set on any XML tree of D. Note that $\emptyset_D$ is definable in any of our XPath fragments (e.g., $\emptyset_D$ can be defined to be A where A is not an element type of D). For fragments $\mathcal{X}$ supporting certain operators, $\mathsf{SAT}(\mathcal{X})$ and the complement problem of $\mathsf{CNT}(\mathcal{X})$ actually coincide. This happens in each of the following two cases:

—The fragment $\mathcal{X}_{(bl,\,[\,],\neg)}$ of *Boolean queries*, that is, queries of the form $\varepsilon[q]$, in any fragment $\mathcal{X}(\ldots, [\,], \neg)$ with negation and qualifiers;

—Any fragment containing negation and closed under the *inverse operator* defined by $\mathsf{inverse}(\downarrow) = \uparrow$, $\mathsf{inverse}(\downarrow^*) = \uparrow^*$ and conversely, $\mathsf{inverse}(\uparrow) = \downarrow$, $\mathsf{inverse}(\uparrow^*) = \downarrow^*$.

These observations are summarized in the following proposition:

PROPOSITION 3.2. (1) *For any XPath fragment* $\mathcal{X}$*, if* $\mathsf{CNT}(\mathcal{X})$ *is in* K *for some complexity class* K*, then* $\mathsf{SAT}(\mathcal{X})$ *is in co*K*. Conversely, if* $\mathsf{SAT}(\mathcal{X})$ *is* K*-hard, then* $\mathsf{CNT}(\mathcal{X})$ *is co*K*-hard.* (2) *For any fragment* $\mathcal{X}_{(bl,\,[\,],\neg)}$ *of Boolean queries, the containment problem* $\mathsf{CNT}(\mathcal{X}_{(bl,\,[\,],\neg)})$ *is reducible in constant time to the complement of the satisfiability problem* $\mathsf{SAT}(\mathcal{X}_{(bl,\,[\,],\neg)})$. (3) *For any fragment* $\mathcal{X}$ *with negation and closed under inverse,* $\mathsf{CNT}(\mathcal{X})$ *is reducible in linear time to the complement of* $\mathsf{SAT}(\mathcal{X})$.

PROOF. (1) is immediate from the comments above. We now show (2). Indeed, for any DTD D and queries $p_1 = \varepsilon[q_1]$, $p_2 = \varepsilon[q_2]$ in $\mathcal{X}_{(bl,\,[\,],\neg)}$, $p_1 \not\subseteq p_2$ under D if and only if (p, D) is satisfiable, where $p = \varepsilon[q_1 \wedge \neg q_2]$, which asserts that there exists an XML tree of D that satisfies p_1 but does not satisfy p_2.

We next show (3). For any DTD D and any queries $p_1, p_2 \in \mathcal{X}$, define a query $p = p_1\big[\neg(\mathsf{inverse}(p_2)[\neg\uparrow])\big]$, where $\mathsf{inverse}(p_2)$ is an extension of the inverse function given above. Intuitively, $T \models p(r, n)$ holds if there exists a node n in T such that $T \models p_1(r, n)$ but $T \not\models p_2(r, n)$, that is, $p_1 \not\subseteq p_2$ under D. That is, there exists a node n reached from the root by following p_1, but the root cannot be reached by tracing back p_2 from n, where the qualifier $[\neg\uparrow]$ conducts the root test. More specifically, $\mathsf{inverse}(p_2)$ is defined as follows, based on the structure of p_2: (1) if $p_2 = l$, then $\mathsf{inverse}(p_2) = \varepsilon[\mathsf{lab}() = l]/\uparrow$; (2) if $p_2 = \downarrow$, then $\mathsf{inverse}(p_2) = \uparrow$; (3) if $p_2 = \downarrow^*$, then $\mathsf{inverse}(p_2) = \uparrow^*$; (4) if $p_2 = \uparrow^*$, then $\mathsf{inverse}(p_2) = \downarrow^*$; (5) if $p_2 = p_3/p_4$, then $\mathsf{inverse}(p_2) = \mathsf{inverse}(p_4)/\mathsf{inverse}(p_3)$; (6) if $p_2 = p_3 \cup p_4$, then $\mathsf{inverse}(p_2) = \mathsf{inverse}(p_3) \vee \mathsf{inverse}(p_4)$; (7) if $p_2 = p_3[q]$,

then $\mathsf{inverse}(p_2) = \varepsilon[q]/\mathsf{inverse}(p_3)$; (8) for all other cases $\mathsf{inverse}(p_2) = p_2$.

To verify that $p_1 \subseteq p_2$ under D if and only if (p, D) is not satisfiable, it suffices to show that for any node n, $T \models p_2(r, n)$ if and only if $T \models \mathsf{inverse}(p_2)[\neg\uparrow](n, r)$. The latter can be verified by a straightforward induction on the structure of p_2. Observe that $\mathsf{inverse}(p_2)$ can be computed in $O(|p_2|)$ time. From these the claim (2) follows immediately.

It should be mentioned that the definition of the $\mathsf{inverse}$ function was first developed in Marx and de Rijke [2005]. □

In Sections 5 and 7, we shall apply Proposition 3.2 to get complexity results for $\mathsf{CNT}(\mathcal{X})$ improving on those in the literature based on the results for $\mathsf{SAT}(\mathcal{X})$ established later on.

3.2. XPath Satisfiability and Normalized DTDs. A mild variant of $\mathsf{SAT}(\mathcal{X})$ for a fragment $\mathcal{X}$ is the problem to determine, given any query $p \in \mathcal{X}$ and any *normalized DTD* D, whether or not there is an XML tree T such that $T \models (p, D)$. Let us refer to this problem as *the satisfiability problem for $\mathcal{X}$ under normalized DTDs*. The next result tells us that for many fragments $\mathcal{X}$, $\mathsf{SAT}(\mathcal{X})$ and the satisfiability problem for $\mathcal{X}$ under normalized DTDs are polynomially equivalent, that is, there are PTIME reductions in both directions.

Proposition 3.3. *For any class $\mathcal{X}$ of XPath queries that allows '$\cup$' (and in addition, label test $\mathsf{lab}() = A$ if $\mathcal{X}$ allows upward modalities), there exists a linear-time function N from DTDs to normalized DTDs, and there exists a function $f : \mathcal{X} \to \mathcal{X}$, computable in $O(|p||D|^3)$ time, such that for any DTD D and any query $p \in \mathcal{X}$, (p, D) is satisfiable if and only if $(f(p), N(D))$ is satisfiable. Moreover, $N(D)$ does not introduce DTD constructs ('+',',','*') not already in D.*

Since the satisfiability problem for $\mathcal{X}$ under normalized DTDs is a special case of $\mathsf{SAT}(\mathcal{X})$, this proposition says that it suffices to consider normalized DTDs when proving either upper or lower bounds for fragments satisfying the restriction above; we shall make frequent use of this in our proofs.

Proof. We first define the function N that maps a general DTD to a normalized DTD. Given a DTD $D = (Ele, Att, P, R, r)$, we define the normalized DTD $N(D) = (Ele', Att', P', R', r')$, which is constructed from D as follows.

For each regular expression on the right-hand side of a production in P, we consider its parse tree and assign new labels A_e for each non-leaf node e in this tree. The label of each leaf node is unchanged. We define Ele' to be the set Ele plus these new labels A_e. The new productions P' can then be straightforwardly determined from these parse trees. Indeed, the operation ('+', ',', '*') in e determines the kind of production, and the (new) labels of the children of e are the element types on the right-hand side of the production. Moreover, we let $r' = r$, $Att' = Att$ and $R' = R$. Hence, it is clear that $N(D) = (Ele', Att', P', R', r')$ is a normalized DTD and can be computed in linear time in the size of D. Note that $N(D)$ does *not* introduce any operators ('+', ',', '*') that do not already exist in D.

Let T be an XML tree such that $T \models D$. We now transform T into T' such that $T' \models N(D)$, following closely the construction of $N(D)$. Let m be a node in T and $\mathsf{children}(m)$ denote the children of m in T. We replace m and its children with a tree T_m rooted at m, in which the leaf nodes are $\mathsf{children}(m)$ (with the corresponding

subtrees of T attached to them), and where the internal nodes are labeled with "new" element types in $Ele' \setminus Ele$. More specifically, let L be the sequence of labels of $\mathsf{children}(m)$. Observe that the production for the label of m in $N(D)$ can be treated as an extended context-free grammar, with the label of m as the start symbol; this grammar is unambiguous by the definition of $N(D)$. Thus, parsing L against $N(D)$ produces a unique parse tree T_m such that (1) the root of T_m is tagged with the label of m, and (2) the leaf nodes of T_m from left to right have a one-to-one correspondence to nodes in $\mathsf{children}(m)$. Finally, we let these leaf nodes carry the same attributes as their corresponding nodes in $\mathsf{children}(m)$. This yields the desired tree T_m. The tree T' can be obtained by applying the transformation above to each node in T starting from the root. Similarly, given $T' \models N(D)$, we can reverse the process above and construct $T \models D$.

Note that T is embedded in T' via a mapping γ that identifies nodes in T with the corresponding nodes in T'. Obviously, γ is an injection.

We next define a rewriting function f such that for any $p \in \mathcal{X}$, $T' \models f(p)$ if and only if $T \models p$. The intuition behind $f(p)$ is that it expands p in such a way that all new nodes in T' are skipped and hence $f(p)$ only "sees" T embedded in T'. Consequently, $f(p)$ will be satisfiable if and only if p is.

In order for $f(p)$ to be able to skip new element types, we need to identify paths in T' consisting of nodes having new element types as labels, that is, those in $Ele' \setminus Ele$. These paths correspond exactly to the paths introduced by the insertion of the trees T_m in the construction of T' as given above. Note that we cannot directly use the trees T_m since we want f to be only dependent on D and not on the instance T'. Therefore, we use $N(D)$ to enumerate all possible paths consisting of new element types. Depending on $\mathcal{X}$, $f(p)$ might have to skip these new paths downwards (if $\mathcal{X}$ contains downward modalities), or upwards (if $\mathcal{X}$ contains upward modalities). Hence, we need two XPath expressions characterizing new downward and upward paths. More specifically, let $A \in Ele$. Then, the production of A in $N(D)$ can be treated as an extended context-free grammar, with A as start symbol. We denote by $\mathsf{newpaths}(A)$ the XPath expression, to be constructed below, characterizing all paths in the parse tree of this grammar such that these paths start from A and consist only of new element types. All such paths are then characterized by the XPath expression $\nabla = \bigcup_{A \in Ele} \mathsf{newpaths}(A)$. Similarly, we denote by $\mathsf{newpaths}^{-1}(A)$ the XPath expression describing all inverse paths in the parse tree such that these paths start from A and consist only of new element types. Again, all such paths are characterized by the XPath expression $\Delta = \bigcup_{A \in Ele} \mathsf{newpaths}^{-1}(A)$. The XPath expressions $\mathsf{newpaths}(A)$ and $\mathsf{newpaths}^{-1}(A)$ are defined inductively depending on the productions in $N(D)$. We first consider $\mathsf{newpaths}(A)$.

—if $P'(A) = B_1, B_2$ or $P'(A) = B_1 + B_2$, then $\mathsf{newpaths}(A) = (B_1/\mathsf{newpaths}(B_1)) \cup (B_2/\mathsf{newpaths}(B_2))$ in case that both B_1 and B_2 are new element types; if only B_1 is a new element type, then $\mathsf{newpaths}(A) = B_1/\mathsf{newpaths}(B_1)$; similar for the case when only B_2 is a new element type;

—if $P(A) = B^*$, then $\mathsf{newpaths}(A) = B/\mathsf{newpaths}(B)$ in case B is a new element type.

We next define $\mathsf{newpaths}^{-1}(A)$. First, we identify all element types C in Ele such that there is a path starting from C and ending at A which solely consists

of new element types. Let $\mathcal{C} \subseteq Ele$ be this set. For each $C \in \mathcal{C}$ we define an XPath expression $\mathsf{invpaths}(A, C)$, characterizing all inverse paths from A to C, inductively on the productions in $N(D)$.

—if $P'(C) = B_1, B_2$ or $P'(C) = B_1 + B_2$, then $\mathsf{invpaths}(A, C) = (\mathsf{invpaths}(A, B_1)[\mathsf{lab}() = B_1] \cup \mathsf{invpaths}(A, B_2)[\mathsf{lab}() = B_2])/\uparrow$ when both B_1, B_2 are new element types from which A can be reached using $N(D)$ with B_1 (respectively, B_2) as start symbol; the case when one or both B_i are old types, or types that do not lead to A, can be dealt with similarly.

—if $P(C) = B^*$, then $\mathsf{invpaths}(A, C) = \mathsf{invpaths}(A, B)[\mathsf{lab}() = B]/\uparrow$ in case that B is a new element type.

Finally, we define $\mathsf{newpaths}^{-1}(A) = \cup_{C \in \mathcal{C}} \mathsf{invpaths}(A, C)$. Observe that '$\cup$', '$\uparrow$' and label tests '$\mathsf{lab}() = A$' are needed to define $\mathsf{newpaths}^{-1}(A)$. The expressions $\mathsf{newpaths}(A)$ and $\mathsf{newpaths}^{-1}(A)$ can be computed in quadratic time in the size $|D|$ of D.

As mentioned above, ∇ (respectively, Δ) is used by $f(p)$ to skip new downward (respectively, upward) paths when $f(p)$ is evaluated on T'. We now define $f(p)$ inductively as follows: (a) if $p = \varepsilon$, then $f(p) = p$; (b) if $p = A$, then $f(p) = \nabla/A$; (c) if $p = \downarrow$, then $f(p) = \cup_{A \in Ele} \nabla/A$; (d) if $p = \downarrow^*$, then $f(p) = \varepsilon \cup \cup_{A \in Ele} \downarrow^*/A$; (e) if $p = \uparrow$, then $f(p) = \uparrow/\Delta$; (f) if $p = \uparrow^*$, then $f(p) = \varepsilon \cup \cup_{A \in Ele} \uparrow^*[\mathsf{lab}() = A]$; (g) if $p = p_1/p_2$, then $f(p) = f(p_1)/f(p_2)$; (h) if $p = p_1 \cup p_2$, then $f(p) = f(p_1) \cup f(p_2)$; (i) if $p = p_1[q]$, then $f(p) = f(p_1)[f(q)]$; (ia) if $q = p_2$, then $f(q) = f(p_2)$; (ib) if q is $\mathsf{lab}() = A$, then $f(q) = q$; (ic) if q is $p_2/@a\ \mathsf{op}\ p_2'/@b$, then $f(q)$ is $f(p_2)/@a\ \mathsf{op}\ f(p_2')/@b$; (id) if q is $p_2/@a\ \mathsf{op}\ `c'$, then $f(q)$ is $f(p_2)/@a\ \mathsf{op}\ `c'$; (ie) if q is $\neg q_1$, then $f(q)$ is $\neg f(q_1)$; similarly for $q_1 \wedge q_2$ and $q_1 \vee q_2$.

Since ∇ and Δ can be computed in $O(|D|^3)$ time, $f(p)$ can be computed in $O(|p||D|^3)$ time. We next verify that f is indeed a function from $\mathcal{X}$ to $\mathcal{X}$ in the following two cases:

—$\mathcal{X}$ contains '$\cup$' but does not contain the upward modalities '$\uparrow$' and '$\uparrow^*$'; or

—$\mathcal{X}$ contains '$\cup$', label tests '$\mathsf{lab}() = A$' and possibly contains upward modalities.

Recall that T is embedded in T' via an (injective) function γ. We now prove that $T' \models f(p)\big(\gamma(n), \gamma(n')\big)$ if and only if $T \models p(n, n')$, by induction on the structure of p.

—$T \models \varepsilon(n, n')$ iff $n = n'$ iff $\gamma(n) = \gamma(n')$ iff $T' \models \varepsilon(\gamma(n), \gamma(n'))$ (γ is an injection);

—$T \models A(n, n')$ iff n' is a child of n and n' is labeled with A iff $\gamma(n')$ is a leaf of $T_{\gamma(n)}$ (as described in the construction of T') and $\gamma(n')$ is labeled with A iff there exists a path in $\mathsf{newpaths}(\gamma(n))$ ending in $\gamma(n')$ and $\gamma(n')$ is labeled with A iff $T' \models \nabla/A(\gamma(n), \gamma(n'))$;

—$T \models \downarrow(n, n')$ iff n' is a child of n iff $\gamma(n')$ is a leaf of $T_{\gamma(n)}$ iff there exists a path in $\mathsf{newpaths}(\gamma(n))$ ending in $\gamma(n')$ iff $T' \models \cup_{A \in Ele} \nabla/A(\gamma(n), \gamma(n'))$;

—$T \models \downarrow^*(n, n')$ iff n' is a descendant of n or $n = n'$ iff $\gamma(n')$ is a descendant of $\gamma(n)$ or $\gamma(n) = \gamma(n')$ iff $T' \models \cup_{A \in Ele} \downarrow^*/A(\gamma(n), \gamma(n'))$ or $T' \models \varepsilon(\gamma(n), \gamma(n'))$;

—$T \models \uparrow(n, n')$ iff n' is the parent of n iff $\gamma(n)$ is a leaf in the tree $T_{\gamma(n')}$ iff there exists a path in $\mathsf{newpaths}^{-1}(\gamma(n))$ ending in n' iff $T \models \cup_{A \in Ele} \Delta/\uparrow[\mathsf{lab}() = A](\gamma(n), \gamma(n'))$;

—$T \models \uparrow^*(n, n')$ iff n' is an ancestor of n or $n = n'$ iff $\gamma(n')$ is an ancestor of $\gamma(n)$ or $\gamma(n) = \gamma(n')$ iff $T' \models \cup_{A \in Ele} \uparrow^*[lab = A](\gamma(n), \gamma(n'))$ or $T' \models \varepsilon(\gamma(n), \gamma(n'))$;

—$T \models p_1/p_2(n, n')$ iff there exists n'' such that $T \models p_1(n, n'')$ and $T \models p_2(n'', n)$ iff (by induction hypothesis) $T' \models f(p_1)(\gamma(n), \gamma(n''))$ and $T' \models f(p_2)(\gamma(n''), \gamma(n'))$ iff $T' \models f(p_1/p_2)(\gamma(n), \gamma(n'))$;

—$T \models (p_1 \cup p_2)(n, n')$ iff $T \models p_1(n, n')$ or $T \models p_2(n, n')$ iff (by induction) $T' \models f(p_1)(\gamma(n), \gamma(n'))$ or $T' \models f(p_2)(\gamma(n), \gamma(n'))$ iff $T' \models f(p_1 \cup p_2)(\gamma(n), \gamma(n'))$;

—$T \models p_1[q](n, n')$ iff $T \models p_1(n, n')$ and $T \models q(n')$ iff (by induction) $T \models f(p_1)(\gamma(n), \gamma(n'))$ and $T' \models f(q)(\gamma(n'))$ iff $T' \models f(p_1[q])(\gamma(n), \gamma(n'))$.

Similarly one can easily verify for any qualifier q in p, $T \models q(n')$ iff $T' \models f(q)(\gamma(n'))$, by observing the fact that γ changes neither the labels nor attributes of nodes in T.

This completes the proof of Proposition 3.3. □

Observe that in the absence of sibling axes, Proposition 3.3 also holds for unordered trees. An unordered tree T satisfies a DTD if for each A element a in T, the list of labels of the children of a is a permutation of some word in $P(A)$, regardless of the ordering of the labels in the list. In this paper we focus on ordered (XML) trees.

4. *Positive XPath Queries*

In this section, we study satisfiability of XPath queries without negation, namely, queries in $\mathcal{X}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \cup, [\ ], =)$, referred to as *positive XPath queries*. We investigate $\mathsf{SAT}(\mathcal{X})$ for various subclasses $\mathcal{X}$ of this fragment.

As observed in Benedikt et al. [2005], positive XPath queries in $\mathcal{X}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \cup, [\ ])$ can be expressed in a fragment of positive existential first-order logic ($\exists^+\mathsf{FO}$) over trees, built up from unary label predicates, binary predicates *child* and *descendant*, and closed under $\wedge$, $\vee$ and $\exists$. A mild (two-sorted) extension of the fragment $\exists^+\mathsf{FO}$ can express queries in $\mathcal{X}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \cup, [\ ], =)$, by supporting unary attribute function, and equality and inequality on attribute values. This fragment does not use universal quantifiers ($\forall$).

Given this existential characterization, it is not surprising that the satisfiability problem is in NP. However, we will find that for even limited positive languages it is NP-hard. We start with a small fragment $\mathcal{X}(\downarrow, \downarrow^*, \cup)$, that is, the downward fragment without qualifiers and data values. We then investigate the impacts of different operators on the complexity of the satisfiability analysis of positive XPath queries, extending $\mathcal{X}(\downarrow, \downarrow^*, \cup)$ by adding qualifiers, upward traversal, recursive axes and data values.

4.1. Downward XPath Queries Without Qualifiers.

We first consider $\mathcal{X}(\downarrow, \downarrow^*, \cup)$ queries. There exists an algorithm that, given any DTD D and any

query p in $\mathcal{X}(\downarrow, \downarrow^*, \cup)$, decides whether or not (p, D) is satisfiable in $O(|p||D|^2)$ time, where $|p|$ and $|D|$ denote the sizes of p and D, respectively. Thus, we have:

THEOREM 4.1. $\mathsf{SAT}(\mathcal{X}(\downarrow, \downarrow^*, \cup))$ *is in PTIME.*

In contrast, recall that the containment problem $\mathsf{CNT}(\mathcal{X}(\downarrow, \downarrow^*, \cup))$ is EXPTIME-complete [Neven and Schwentick 2006]. This shows that the satisfiability analysis is quite different from its containment counterpart.

PROOF. We provide a decision algorithm that, given a query p in $\mathcal{X}(\downarrow, \downarrow^*, \cup)$ and a DTD D, decides whether or not (p, D) is satisfiable in $O(|p||D|^5)$ time. By Proposition 3.3, we may assume that D is a normalized DTD.

Before we get to the decision algorithm, we explain the notion of DTD graphs. Consider a normalized DTD $D = (Ele, Att, P, R, r)$. The DTD graph $G_D = (V, E, \lambda)$ of D is a directed labeled graph rooted at r, where $V = Ele \cup Att$, and $(v, w) \in E$ if w appears in $P(v)$ or $w \in R(v)$. In case there is an edge (v, w) because w appears in $P(v)$, we define $\lambda(v, w) \in \{\vee, \wedge, *\}$ depending on what kind of production $P(v)$ is (disjunction, concatenation, Kleene star).

The decision algorithm employs the parse-tree representation of p. We start by compiling the list $\mathcal{L}$ of all subqueries of p, topologically ordered such that p_1 precedes p_2 in $\mathcal{L}$ if p_1 is a subquery of p_2. For each $p' \in \mathcal{L}$ and element type A in D we use a variable $\mathsf{reach}(p', A)$ to hold the set of all element types reachable from A via p' in the DTD graph G_D. These variables are initially set to the empty set $\emptyset$ and are computed based on dynamic programming. More specifically, the algorithm works as follows:

(1) For each $p' \in \mathcal{L}$ (in the order of $\mathcal{L}$) and element type A in D, we compute $\mathsf{reach}(p', A)$ depending on the structure of p'.
   (a) $p' = \varepsilon$: then $\mathsf{reach}(p', A) = \{A\}$;
   (b) $p' = l$: $\mathsf{reach}(p', A) = \{l\}$ if $P(A)$ contains l, and $\mathsf{reach}(p', A) = \emptyset$ otherwise;
   (c) $p' = \downarrow$: then $\mathsf{reach}(p', A)$ is the set of subelement types in $P(A)$;
   (d) $p' = \downarrow^*$: then $\mathsf{reach}(p', A)$ is the set of element types in G_D reachable from A;
   (e) $p' = p_1 \cup p_2$: then $\mathsf{reach}(p', A) = \mathsf{reach}(p_1, A) \cup \mathsf{reach}(p_2, A)$;
   (f) $p' = p_1/p_2$: then $\mathsf{reach}(p', A) = \cup_{B \in \mathsf{reach}(p_1, A)} \mathsf{reach}(p_2, B)$.
(2) Return $\mathsf{reach}(p, r) \neq \emptyset$, where r is the root type of D.

The algorithm iterates over all subqueries in $\mathcal{L}$ and all element types in Ele. Hence, the main loop in the algorithm is executed at most $O(|p||D|)$ times. Each step in the loop takes at most $O(|D|)$ time, for example, the case $p' = p_1/p_2$. Hence, the worst-case time complexity is $O(|p||D|^2)$. By Proposition 3.3, the normalization step also takes $O(|p||D|^3)$ time. Moreover, the normalization step rewrites p into an equivalent query of size $O(|p||D^3|)$ on the normalized DTD. This brings the overall worst-case time complexity to $O(|p||D|^5)$.

We now prove the correctness of the algorithm, that is, the algorithm returns true iff (p, D) is satisfiable. Suppose that there is a tree $T \models (p, D)$. Then it is easy to verify by induction on the structure of p and the semantics of XPath queries that $\mathsf{reach}(p, r) \neq \emptyset$. More specifically, we show by induction that for any subquery p' of p and any A element n of T, if there exists a B element n' of T such that $T \models p'(n, n')$, then $B \in \mathsf{reach}(p', A)$. As an example, consider $T \models p_1/p_2(n, n')$. Then

there exists a node n'' of T such that $T \models p_1(n, n'')$ and $T \models p(n'', n')$. Let C be the tag of n''. By the induction hypothesis, $C \in \mathsf{reach}(p_1, A)$ and $B \in \mathsf{reach}(p_2, C)$. By the processing of p_1/p_2 in the algorithm, we have that $B \in \mathsf{reach}(p', A)$. All other cases can be verified similarly.

Conversely, if the algorithm returns true, we construct an XML tree $\mathsf{Tree}(p, D)$ that satisfies both p and D. More specifically, for each subquery p' of p in $\mathcal{L}$ and pairs of element types A, B in *Ele*, we first define a path $\mathsf{path}(p', A, B)$ in G_D starting in A and ending in B such that $B \in \mathsf{reach}(p', A)$. We then construct $\mathsf{Tree}(p, D)$ using these paths. The construction of $\mathsf{path}(p', A, B)$ is computed as follows: (a) $p' = \varepsilon$, then $\mathsf{path}(p', A, B) = \varepsilon$ if $A = B$ and is the empty set $\emptyset$ otherwise; (b) $p' = l$, then $\mathsf{path}(p', A, B) = l$ if $B \in \mathsf{reach}(p', A)$ and is $\emptyset$ otherwise; (c) $p' = \downarrow$, then $\mathsf{path}(p', A, B) = B$ if $B \in \mathsf{reach}(p', A)$ and is $\emptyset$ otherwise; (d) $p' = \downarrow^*$, then $\mathsf{path}(p', A, B)$ is a shortest path in G_D from A to B (excluding A) if $B \in \mathsf{reach}(p', A)$ and is $\emptyset$ otherwise; (e) $p' = p_1 \cup p_2$, then $\mathsf{path}(p', A, B) = \mathsf{path}(p_1, A, B)$ if $\mathsf{path}(p_1, A, B)$ is nonempty and $\mathsf{path}(p', A, B) = \mathsf{path}(p_2, A, B)$ otherwise; and (f) $p' = p_1/p_2$, then $\mathsf{path}(p', A, B)$ is the concatenation of $\mathsf{path}(p_1, A, C)$ and $\mathsf{path}(p_2, C, B)$ if $B \in \mathsf{reach}(p', A)$ and $C \in \mathsf{reach}(p_1, A)$ such that $B \in \mathsf{reach}(p_2, C)$; $\mathsf{path}(p', A, B)$ is $\emptyset$ otherwise.

Since the algorithm returns true, there must exist some $B \in \mathsf{reach}(p, r)$ such that $\mathsf{path}(p, r, B)$ is nonempty. We construct $\mathsf{Tree}(p, D)$ as follows. Initially, $\mathsf{Tree}(p, D)$ consists of a root node labeled with r from which a chain of nodes emanates such that the labels of the nodes in this chain are precisely $\mathsf{path}(p, r, B)$, that is, the chain corresponds to $\mathsf{path}(p, r, B)$ from r to B in G_D. Next, by using productions of the DTD D, we expand the tree into a finite XML tree conforming to D (we assume that all element types in D are terminating.) By the semantics of XPath, it is easy to verify that $\mathsf{Tree}(p, D) \models p$. □

In contrast to Theorem 4.1, in Theorem 6.11, we shall prove that queries in $\mathcal{X}(\downarrow, \downarrow^*, \cup, [\,])$ are always satisfiable in the absence of label tests and DTDs.

Obviously, this is an extreme case where the presence of DTDs complicates the satisfiability analysis. Further complexity results for the satisfiability problem in the absence of DTDs will be presented in Section 6.

4.2. DOWNWARD XPATH QUERIES WITH QUALIFIERS. We now study $\mathcal{X}(\downarrow, \downarrow^*, \cup, [\,])$, that is, the extension of $\mathcal{X}(\downarrow, \downarrow^*, \cup)$ by adding qualifiers. The result below shows that adding qualifiers does make our lives harder: the satisfiability problem becomes intractable, even in the absence of recursion ($\downarrow^*$), and without either disjunction (union $\cup$) or wildcard ($\downarrow$).

PROPOSITION 4.2. *The following problems are NP-hard:* (1) $\mathsf{SAT}(\mathcal{X}(\downarrow, [\,]))$*;* (2) $\mathsf{SAT}(\mathcal{X}(\cup, [\,]))$.

PROOF.

(1) We show that $\mathsf{SAT}(\mathcal{X}(\downarrow, [\,]))$ is NP-hard by reduction from the 3SAT problem. An instance of this problem consists of a well-formed Boolean formula $\phi = C_1 \wedge \cdots \wedge C_n$ of which we want to decide satisfiability. The 3SAT problem is known to be NP-complete (cf. Papadimitriou [1994]). Assume that the variables in ϕ are $x_1, \ldots, x_m$.

Given ϕ, we define a DTD D and a query $\mathsf{XP}(\phi)$ in $\mathcal{X}(\downarrow, [\,])$ such that ϕ is satisfiable iff $(\mathsf{XP}(\phi), D)$ is satisfiable. We define the DTD $D = (\mathit{Ele}, \mathit{Att}, P, R, r)$

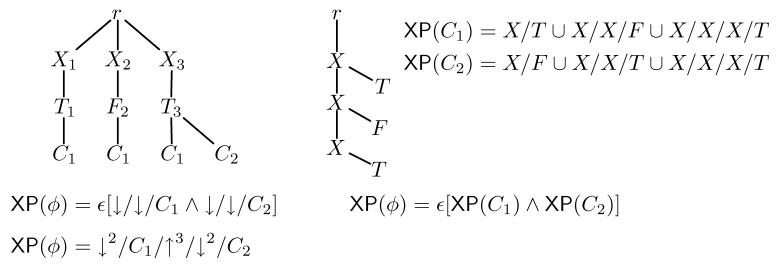

FIG. 1. Example encoding of a 3SAT instance ϕ in the proofs of Propositions 4.2 and 4.3. Left: an XML tree conforming to the DTD defined in the reduction for $\mathcal{X}(\downarrow, [\,])$ and $\mathcal{X}(\downarrow, \uparrow)$. Right: reduction for $\mathcal{X}(\cup, [\,])$.

as follows:

$$
\begin{array}{ll}
\multicolumn{2}{l}{Ele = \{X_j, T_j, F_j \mid j \in [1,m]\} \cup \{C_i \mid i \in [1,n]\} \cup \{r\}.} \\
P: & r \to X_1, \ldots, X_m, \qquad X_j \to T_j + F_j, \\
& T_j \to C_{j_1}, \ldots, C_{j_k} \qquad \text{/* all } C_{j_i} \text{ in which } x_j \text{ appears */} \\
& F_j \to C_{j_1}, \ldots, C_{j_k}. \qquad \text{/* all } C_{j_i} \text{ in which } \bar{x}_j \text{ appears */} \\
\multicolumn{2}{l}{Att = \emptyset,\ R(A) = \emptyset \text{ for all } A \in Ele.}
\end{array}
$$

Furthermore, we define the $\mathcal{X}(\downarrow, [\,])$ query $\mathsf{XP}(\phi) = \varepsilon[\downarrow/\downarrow/C_1 \wedge \cdots \wedge \downarrow/\downarrow/C_n]$. An illustration of this reduction is given in Figure 1 (left). Then, there exists a finite XML tree T such that $T \models (\mathsf{XP}(\phi), D)$ iff ϕ is satisfiable. Indeed, there is a one-to-one correspondence between XML trees T of D on the one hand, and valid truth assignments for the x_i variables in ϕ on the other hand. Moreover, $T \models \mathsf{XP}(\phi)$ iff all clauses appear as the leaf nodes of the tree T. This in turn is the case iff the truth assignment corresponding to T makes all clauses true and hence is a solution of the 3SAT problem.

(2) We show that $\mathsf{SAT}(\mathcal{X}(\cup, [\,]))$ is NP-hard by reduction from the 3SAT problem. Given an instance $\phi = C_1 \wedge \cdots \wedge C_n$ of 3SAT, we define the DTD $D = (Ele, Att, P, R, r)$:

$$
\begin{array}{l}
Ele = \{r, X, T, F\}. \\
P: \quad r \to X, \quad X \to (X + \varepsilon), (T + F). \quad Att = \emptyset,\ R(A) = \emptyset \text{ for all } A \in Ele.
\end{array}
$$

As illustrated in Figure 1 (right), an XML tree of D consists of a chain of X elements. Each X element has either a T or an F child, which ensures that each variable x_i has a unique truth value. Here, x_i is encoded by X^i, the chain $X/\cdots/X$ of length i.

Furthermore, we encode ϕ in terms of a query $\mathsf{XP}(\phi) = \varepsilon[\mathsf{XP}(C_1) \wedge \cdots \wedge \mathsf{XP}(C_n)]$ in $\mathcal{X}(\cup, [\,])$, where $\mathsf{XP}(C_i)$ is defined as follows.

—Encoding variables: for each variable x_i in ϕ let $\mathsf{XP}(x_i) = X^i/T$ and $\mathsf{XP}(\bar{x}_i) = X^i/F$.

—Encoding clauses: For each clause C_j we let $\mathsf{XP}(C_j)$ be C_j in which each x_i is replaced by $\mathsf{XP}(x_i)$ and each $\bar{x}_i$ is replaced by $\mathsf{XP}(\bar{x}_i)$; e.g., if $C = x_i \vee \bar{x}_j \vee x_k$, then $\mathsf{XP}(C) = X^i/T \cup X^j/F \cup X^k/T$. It is clear that a tree of D satisfies $\mathsf{XP}(C)$ iff C is satisfiable.

It is easy to verify that ϕ is satisfiable iff there exists an XML tree $T \models (\mathsf{XP}(\phi), D)$. □

4.3. UPWARD XPATH QUERIES WITHOUT QUALIFIERS. Alternatively we extend $\mathcal{X}(\downarrow, \downarrow^*, \cup)$ by allowing upward modalities. The presence of upward modalities also complicates the satisfiability analysis: the satisfiability problem also becomes intractable, even in the absence of recursion ($\downarrow^*$, $\uparrow^*$), union ($\cup$) and qualifiers ([ ]).

PROPOSITION 4.3. $\mathsf{SAT}(\mathcal{X}(\downarrow, \uparrow))$ *is NP-hard.*

PROOF. We show that $\mathsf{SAT}(\mathcal{X}(\downarrow, \uparrow))$ is NP-hard by reduction from the 3SAT problem. An illustration of this reduction is depicted in Figure 1 (left). Given an instance $\phi = C_1 \wedge \cdots \wedge C_n$ of 3SAT, we define the DTD $D = (Ele, Att, P, R, r)$ to be the same as the one used in the proof of Proposition 4.2 (1) given above. We define the $\mathcal{X}(\downarrow, \uparrow)$ query $\mathsf{XP}(\phi) = \downarrow^2/C_1/\uparrow^3/\downarrow^2/C_2/\uparrow^3/\cdots\downarrow^2/C_n$, where $\downarrow^2 = \downarrow/\downarrow$ and $\uparrow^3 = \uparrow/\uparrow/\uparrow$. Then, it can be easily verified that there exists an XML tree T such that $T \models (\mathsf{XP}(\phi), D)$ iff all clauses appear as the leaf nodes of the tree, or in other words, ϕ is satisfiable. □

4.4. ADDING RECURSION AND DATA VALUES. For positive XPath queries with qualifiers, the presence of recursion and data values does not increase the complexity of the satisfiability analysis. Indeed, below we show that adding recursion and data values does not move the problem beyond NP. In contrast, it has been shown [Neven and Schwentick 2006; Wood 2002] that in the presence of DTDs, the containment problem $\mathsf{CNT}(\mathcal{X})$ is EXPTIME-hard when $\mathcal{X}$ is either $\mathcal{X}(\downarrow^*, \cup)$ or $\mathcal{X}(\downarrow, \downarrow^*, [\,])$; and it is in EXPTIME for $\mathcal{X}(\downarrow, \downarrow^*, \cup, [\,])$. Neither XPath's data value equality nor upward modalities are considered in Neven and Schwentick [2006], and Wood [2002].

THEOREM 4.4. $\mathsf{SAT}(\mathcal{X}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \cup, [\ ], =))$ *is in NP. Furthermore,* $\mathsf{SAT}(\mathcal{X})$ *is NP-complete for any fragment* $\mathcal{X}$ *of* $\mathcal{X}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \cup, [\,], =)$ *that supports either* $(\downarrow, [\,])$*, or* $(\cup, [\,])$*, or* $(\downarrow, \uparrow)$.

PROOF. We prove the NP upper bound for $\mathsf{SAT}(\mathcal{X}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \cup, [\ ], =))$. Combined with the NP lower bound given in Propositions 4.2 and 4.3, this proves NP-completeness for the fragments in the statement of the theorem.

First, observe that for any $p \in \mathcal{X}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \cup, [\,], =)$ we may assume that p does not contain $\cup$ or $\vee$. Indeed, for each occurrence of $\cup$ or $\vee$ in p, we can guess nondeterministically one of the alternatives. So we consider $p \in \mathcal{X}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, [\,], =)$. Let D be a DTD. The proof consists of the following steps. First, we define a graph representation of p, called the *skeleton* T_p of p. Next, we show that there exists a tree T such that $T \models p$ if and only if there exists an *embedding* γ of the skeleton T_p of p in T. We call $\gamma(T_p)$ a *witness skeleton* for p in T. We then prove the main lemma, which states that, for a tree $T \models D$, if there exists a witness skeleton for p in T, then there exists a tree $T' \models D$ and a witness skeleton for p in T' whose depth is polynomial in the size of p and D. The proof of this lemma is based on a *short-cutting technique*. Once we established the bound on the depth of the witness skeletons for p, we can guess a candidate witness skeleton for p based on D alone and check whether it is a real witness skeleton in PTIME.

4.4.1. *Skeletons and Witnesses.* For a given p, we define the skeleton T_p inductively on the structure of p. A skeleton is a graph obtained by the construction

below, in which nodes can be either unlabeled or can have an element or attribute label, and in which edges can be either unlabeled, or can have a label from the set $\{\downarrow, \uparrow, \downarrow^*, \uparrow^*, =, \neq\}$. Moreover, a skeleton has a unique root node, which is labeled with r. For the purpose of defining a skeleton, we also identify a unique "exit node" labeled with e (the exit node will be the node to which earlier computed skeletons can be attached). The exit node can be any node in the skeleton. We define the skeleton T_p for p as follows:

—If $p = \varepsilon$, then $T_p = (\{n\}, \emptyset)$ is the skeleton consisting of a single node n labeled with r. The exit node of T_p is n.

—If $p = A$, then $T_p = (\{n_1, n_2\}, \{(n_1, n_2)\})$ consists of two nodes and a single edge, where n_1 is labeled with r and n_2 is labeled with A. The edge is labeled with $\downarrow$, and the exit node of T_p is n_2.

—If $p = \downarrow$, then $T_p = (\{n_1, n_2\}, \{(n_1, n_2)\})$, where n_1 is labeled with r and n_2 is unlabeled. The edge is labeled with $\downarrow$, and the exit node is n_2.

—If $p = \downarrow^*$, then $T_p = (\{n_1, n_2\}, \{(n_1, n_2)\})$, where n_1 is labeled with r, n_2 is unlabeled and the edge is labeled with $\downarrow^*$. The exit node of T_p is n_2.

—If $p = \uparrow$ or $p = \uparrow^*$, then the construction of T_p is similar to the $\downarrow$, $\downarrow^*$ cases but using $\uparrow$ and $\uparrow^*$ as labels instead.

—If $p = p_1/p_2$, then T_p is obtained by identifying the root node n_2 of T_{p_2} with the exit node e_1 of T_{p_1}, keeping the label of e_1 while removing the label r of n_2. The root node of T_p is the root node of T_{p_1} and the exit node of T_p is the exit node of T_{p_2}.

—If $p = p_1[q]$, then T_p is obtained by identifying the exit node e_1 of T_{p_1} with the root node n_q of T_q, keeping the label of e_1 while removing the label r of n_q. Moreover, the root node of T_p is the root node of T_{p_1} and the exit node of T_p is the exit node of T_{p_1}.

—The skeleton for qualifiers q is defined as follows:

  —If $q = p'$, then T_q is the skeleton $T_{p'}$.

  —If $q =$ '$\mathsf{lab}() = A$', then $T_q = (\{n\}, \emptyset)$ in which n is labeled with A. Moreover, this node is both the root and exit node of T_q.

  —If $q = p_1/@a$ op 'c', with $\mathsf{op} \in \{=, \neq\}$, then T_q is obtained by adding an edge (e_1, n) from the exit node e_1 of T_{p_1}, where n is labeled with $@a$ op 'c'. The root and exit node of T_q are both the root node of T_{p_1}.

  —If $q = p_1/@a$ op $p_2/@b$, with $\mathsf{op} \in \{=, \neq\}$, then T_q is obtained by identifying the root nodes of T_{p_1} and T_{p_2} and adding edges (e_1, n_1), (e_2, n_2), (n_1, n_2), where e_i is the exit node of T_{p_i}, n_1 is labeled with $@a$ and n_2 is labeled with $@b$. The edge (n_1, n_2) is labeled with op. The root node and exit node of T_q are the root of T_{p_1}.

  —If $q = q_1 \wedge q_2$, then T_q is obtained by identifying the root nodes of T_{q_1} and T_{q_2}. The root and exit node of T_q are the root node of T_{q_1}.

Having defined the skeleton of an XPath expression, we explain what it means to have a witness skeleton for p in an XML tree. Let T be an XML tree and T_p be the skeleton of p. We say that there *exists a witness skeleton for p in T* if there exists a mapping $\gamma : T_p \to T$ satisfying the following conditions: (a) $\gamma(r)$ is the root of T; (b) for any node $n \in T_p$, $\gamma(n)$ has the same label (if specified) as n; (c) $\gamma(n_2)$ is a child of $\gamma(n_1)$ in T if (n_1, n_2) is a $\downarrow$-labeled edge in T_p; (d) $\gamma(n_2)$ is the parent

of $\gamma(n_1)$ in T if (n_1, n_2) is a $\uparrow$-labeled edge in T_p; (e) $\gamma(n_2)$ is a descendant in of $\gamma(n_1)$ in T if (n_1, n_2) is an $\downarrow^*$-labeled edge in T_p; (f) $\gamma(n_2)$ is an ancestor of $\gamma(n_1)$ in T if (n_1, n_2) is an $\uparrow^*$-labeled edge in T_p; (g) $\gamma(n_1)$ op $\gamma(n_2)$, with $\mathsf{op} \in \{=, \neq\}$, if there is an edge between n_1 and n_2 labeled with op.

If such an embedding γ exists, we denote this by $T_p \subseteq_\gamma T$. It follows directly from the semantics of XPath that $T \models p$ if and only if there exists a γ such that $T_p \subseteq_\gamma T$.

4.4.2. *Shortcutting.* Let T be a tree such that $T \models D$ and there exists a witness skeleton for p in T. Let γ be the embedding. Define the *depth of a node* in a witness skeleton for p in T as the depth of this node in T. The *depth of a witness skeleton* for p in T is the depth of the deepest node of $\gamma(T_p)$ in T. We now prove the following:

LEMMA 4.5. *If $T \models D$ and there exists a witness skeleton for p in T, then there exists a tree $T' \models D$ with a witness skeleton for p in T' of depth bounded by* $(3|p| - 1)|D|$.

Assuming this lemma, we can conclude the proof of the NP upper bound. Let T be a tree containing a witness skeleton for p. A witness skeleton for p in T induces a subtree of T that includes all the paths from the root of T to the deepest nodes in the witness skeleton. We call this tree a *witness tree for p in T*. We guess a witness tree for p without actually generating the bigger tree T in which it resides. The number of branches in any witness tree is bounded by the number of branches in the skeleton (here we ignore the op-labeled edges in T_p), which is bounded by the number of subqualifiers in p. Moreover, by Lemma 4.5, we know that it is sufficient to look at witness skeletons (and hence witness trees) of bounded depth. It is therefore sufficient for our nondeterministic algorithm to guess a candidate tree $\mathsf{witness}(p, D)$ consisting of at most $|p|$ branches of length at most $(3|p| - 1)|D|$ whose nodes are labeled with element types in D. We then need to validate that $\mathsf{witness}(p, D)$ can be expanded to a tree conforming to the DTD. For this, we check for each node n in $\mathsf{witness}(p, D)$ the following: Let $\mathsf{children}(n)$ be the list of nodes of children of n and A be the tag of n. Then, the labels of $\mathsf{children}(n)$ must be a substring satisfying the production rule of A in the DTD (recall that we assume that D consists of terminating nodes only.) For each node in $\mathsf{witness}(p, D)$, this can clearly be checked in time $O(|\mathsf{children}(n)|\,|D|)$. If all nodes in $\mathsf{witness}(p, D)$ pass this test, then $\mathsf{witness}(p, D)$ expands to a tree conforming to D. Hence, this test can be conducted in time $O(|\mathsf{witness}(p, D)||D|) \leq O((3|p|^2 - |p|)|D|^2)$.

We next prove Lemma 4.5. We first introduce some notations. Let $T \models D$ contain a witness skeleton for p. A path $\Gamma(v_1, v_2)$ in T (starting from a node v_1 and ending at a node v_k) is a sequence of nodes $v_1, v_2, \ldots, v_{k-1}, v_k$ such that (v_i, v_{i+1}) is an edge in T for $i \in [1, k - 1]$. For each node v in T, we define $\mathsf{cover}(v) = \{(n_1, n_2) \mid (n_1, n_2)$ is a $\downarrow^*$-labeled edge in $T_p, v \in \Gamma(\gamma(n_1), \gamma(n_2))\}$.

Let v_k be a node in the witness skeleton for p in T with a maximal depth. Let $\rho = r, v_1, \ldots, v_k$ be the path in T from the root r to v_k. We can partition ρ as follows: v_i is in the same part as v_{i+1} iff $\mathsf{cover}(v_i) \neq \emptyset$ and $\mathsf{cover}(v_i) = \mathsf{cover}(v_{i+1})$. If $\mathsf{cover}(v_i) = \emptyset$, then v_i belongs to the part $\{v_i\}$. We denote by ρ_0 the set of nodes v in ρ for which $\mathsf{cover}(v) = \emptyset$. Note that all parts of this partition consists of a path segment in ρ.

We now show how to size down ρ and in this process change T into T' such that $T_p \subseteq_\gamma T'$ and T' still conforms to D. Repeating the sizing down process we end

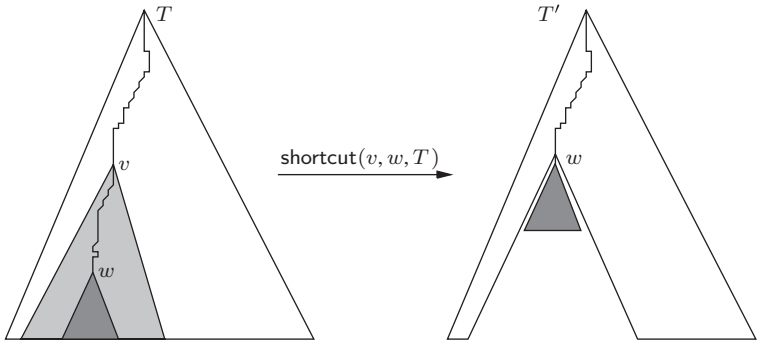

FIG. 2. Shortcut operation.

up with a tree T' that has a witness skeleton for p in T' of depth no larger than $(3|p|-1)|D|$. Of course, if $|\rho| \leq (3|p|-1)|D|$, we do not have to do anything. So, suppose that $|\rho| > (3|p|-1)|D|$. Let us concentrate on a part $\tau = v_s, v_{s+1}, \ldots, v_t$ in the partition of ρ as defined above. There are two kinds of nodes in τ: the set of nodes that are in the witness skeleton, denoted by by W_τ; and nodes that are not, for which the cover is always nonempty. We chop parts of the tree T involving nodes not in W_τ by means of the cut operation:

shortcut(v, w, T): Let w be a descendant of v in T, T_w be the subtree of T rooted at w and let T_v be the subtree of T rooted at v. Then the result of shortcut(v, w, T) is the tree obtained by replacing T_v by T_w in T, as illustrated in Figure 2.

Let $v_i, v_{i+1}, \ldots, v_{i+\ell}$ be a path segment of maximal length in τ such that all nodes are not in W_τ. Now, as long as there are more than $|D|$ nodes between v_i and $v_{i+\ell}$, there must be two nodes v_{i_1} and v_{i_2} with the same element type in τ. We apply shortcut(v_{i_1}, v_{i_2}, T) until no such two nodes exist anymore. After we have done all the shortcuts between v_i and $v_{i+\ell}$, we denote the resulting tree by T'.

We claim that such shortcut operation does not affect the embedding γ. Clearly, shortcutting leaves γ unchanged for witness skeleton nodes in $T \setminus T_{v_i}$ and $T_{v_{i+\ell}}$. Also up-and-downward relations are preserved between any such nodes. We now show that $T_{v_i} \setminus T_{v_{i+\ell}}$ does not contain any witness skeleton nodes and hence does not affect γ. Indeed, suppose that there exists a node w in the witness skeleton for p in T that is located in a subtree T_{v_j} rooted at one of the nodes $v_i, v_{i+1}, \ldots, v_{i+\ell}$, say v_j for some $i \leq j < i + \ell$. Then there is a chain of witness skeleton nodes starting from the root to the witness skeleton node w such that consecutive witness skeleton nodes in this chain are images under the embedding γ of nodes connected by an edge in the skeleton T_p (this holds for any witness skeleton node). Then, there must be another witness skeleton node w_1 that is not in the subtree T_{v_j}. In other words, the path in T from w_1 to w intersects τ. Assume that this intersection is $v_i, v_{i+1}, \ldots, v_j, v'_{j+1}, \ldots, w$. Then, by the definition of γ, there must exist a node w_2 in the segment $v'_{j+1}, \ldots, w$ such that (w_1, w_2) is the image under γ of a $\downarrow^*$-labeled edge e in T_p, because v_j is not a witness skeleton node. From this, it follows that cover(v_j) is different from cover(v_{j+1}), since e is in cover(v_j) but not in cover(v_{j+1}). However, this contradicts the assumption that v_j and v_{j+1} are in the same equivalent class τ in the partition of ρ. From the argument above, it

follows that $T_p \subseteq_\gamma T'$. Moreover, since shortcutting replaces nodes with the same element type and attributes, one can verify that $T' \models D$.

We process each path segment of maximal length of nodes not in W_τ as described above. Since there are at most $|W_\tau| + 1$ such path segments, after the shortcut, τ has length at most $(|W_\tau| + 1)|D|$. Moreover, we can do this for each part τ in the partition of ρ with nonempty cover and in this way obtain that, after the shortcut, ρ itself has length at most $|\rho_0| + \sum_{\tau \subseteq (\rho \setminus \rho_0)} (|W_\tau| + 1)|D| \leq (|W_\rho| + \#\text{parts}(\rho \setminus \rho_0))|D|$, where ρ_0 is the set of nodes in ρ that have a empty cover, W_ρ is the set of witness skeleton nodes along ρ and $\#\text{parts}(\rho \setminus \rho_0)$ denotes the number of parts τ in the partition of ρ with a nonempty cover.

We now prove an upper bound on $\#\text{parts}(\rho \setminus \rho_0)$ in terms of the size of p:

LEMMA 4.6. *Let $T_p \subseteq_\gamma T$. If p has k occurrences of $\downarrow^*$, then the partition defined above on paths ρ in T from the root of T to a node of maximal depth in $\gamma(T_p)$ consists of at most $2k - 1$ parts corresponding to nonempty covers.*

PROOF. Before we prove the upper bound, we define an ordering $\leq$ on the set V of $\downarrow^*$-labeled edges in T_p, or equivalently the occurrences of $\downarrow^*$ in p. Let ρ be a path in T as specified in the lemma. Without loss of generality, we assume that $\Gamma(\gamma(n_1), \gamma(n_2)) \subseteq \rho$ for all $(n_1, n_2) \in V$. We then define $(n_1, n_2) \leq (n_1', n_2')$ if $\gamma(n_1)$ appears before or at the same position as $\gamma(n_1')$ in ρ. Based on $\leq$, we can define (n_1^i, n_2^i) as the ith edge in V.

Define $\mathsf{cover}_i(v) = \{(n_1^j, n_2^j) \mid v \in \Gamma(\gamma(n_1^j), \gamma(n_2^j)),\ j \leq i\}$. For $i > 0$, let ρ_i be the nodes in ρ for which $\mathsf{cover}_i(v) \neq \emptyset$. For $i > 0$, we partition ρ_i similarly as before using cover_i. We denote by $\#\text{parts}(\rho_i)$ the number of parts in this partition of ρ_i with nonempty covers. We now show that $\#\text{parts}(\rho_i) \leq 2i - 1$ for $i \in [1, k]$ by induction on i.

Base Case: When $i = 1$, clearly $\rho_1 = \Gamma(\gamma(n_1^1), \gamma(n_2^1))$ and hence $\#\text{parts}(\rho_1) \leq 1$.

Induction Step: Suppose that $\#\text{parts}(\rho_{i-1}) \leq 2(i-1) - 1$ and consider $\Gamma(\gamma(n_1^i), \gamma(n_2^i))$. By the definition of $\leq$, either (1) $\Gamma(\gamma(n_1^i), \gamma(n_2^i))$ does not intersect ρ_{i-1}; or (2) there exists an ℓ, $1 \leq \ell < i$ such that $\Gamma(\gamma(n_1^i), \gamma(n_2^i))$ intersects the path fragment $\Gamma' = \cup_{j=\ell}^{i-1} \Gamma(\gamma(n_1^j), \gamma(n_2^j))$, and this path fragment is disjoint with any $\Gamma(\gamma(n_1^j), \gamma(n_2^j))$ for $j < \ell$. Moreover, $\Gamma(\gamma(n_1^i), \gamma(n_2^i)) \cap \Gamma(\gamma(n_1^j), \gamma(n_2^j)) = \emptyset$ for $j < \ell$. In the case (1), ρ_i consists of ρ_{i-1} plus the new (disjoint) part $\Gamma(\gamma(n_1^i), \gamma(n_2^i))$. Hence, $\#\text{parts}(\rho_i) = \#\text{parts}(\rho_{i-1}) + 1 < 2i - 1$. In the case (2), we distinguish between the following cases depending on the intersection of $\Gamma(\gamma(n_1^i), \gamma(n_2^i))$ with Γ':

—$\Gamma' \subseteq \Gamma(\gamma(n_1^i), \gamma(n_2^i))$: then at most one new part in ρ_i is created after Γ';

—$\Gamma' \supset \Gamma(\gamma(n_1^i), \gamma(n_2^i))$: then at most two parts in ρ_{i-1} are split into two parts;

—Γ' and $\Gamma(\gamma(n_1^i), \gamma(n_2^i))$ overlap but neither one is contained in the other: then at most one part in ρ_{i-1} is split into two parts and at most one new part after Γ' is created.

Since all these cases introduce at most two extra parts in ρ_i, we have that $\#\text{parts}(\rho_i) \leq \#\text{parts}(\rho_{i-1}) + 2 = 2i - 1$. Since $\rho_k = \rho \setminus \rho_0$, we obtain the desired upper bound. This concludes the proof of Lemma 4.6 □

To conclude the proof of Lemma 4.5, observe that Lemma 4.6 tells us that for each path ρ, $|\rho| < (|W_\rho| + \#\text{parts}(\rho \setminus \rho_0))|D| < (3|p| - 1)|D|$. Thus, shortcuting these paths leads to tree T' that contains a witness skeleton for p of depth at most $(3|p| - 1)|D|$. □

5. *XPath Fragments with Negation*

In this section, we show that allowing negation in qualifiers makes the satisfiability analysis different in nature. With negation, one must deal with both universal and existential quantifiers. In contrast to positive XPath, adding data values and/or recursion to XPath fragments with negation has an enormous effect: with recursive axes, adding data values makes the satisfiability problem undecidable, while without recursion there is a jump to NEXPTIME. That is, the interaction between recursion, data values and negation is rather intricate.

Most previous work on XPath containment/satisfiability bounds [Deutsch and Tannen 2005; Hidders 2004; Lakshmanan et al. 2004; Miklau and Suciu 2004; Neven and Schwentick 2006; Wood 2002] has considered either no negation or restricted negation. From the EXPTIME lower bounds on containment in Neven and Schwentick [2006] and Wood [2002] plus Proposition 3.2, we get an EXPTIME lower bound for $\mathsf{SAT}(\mathcal{X}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \cup, [\,], \neg))$. Marx [2004] considers an extension of XPath that subsumes $\mathcal{X}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \cup, [\,], \neg)$, and proves an EXPTIME upper bound via reduction to propositional dynamic logic. We include this result here for completeness.

We first study the impact of negation by investigating $\mathsf{SAT}(\mathcal{X}(\downarrow, [\,], \neg))$. We then gradually extend $\mathcal{X}(\downarrow, [\,], \neg)$ by adding upward modalities, recursion, and data values, investigating the impact of these operators on the satisfiability analysis in the presence of negation.

5.1. Nonrecursive XPath Queries with Negative Qualifiers.

We first consider fragments of XPath with negation but without recursion. The result below tells us three things. First, the presence of negation makes the satisfiability analysis PSPACE-hard (Proposition 5.1). Second, the bound is tight (Theorem 5.2). Third, in contrast to Propositions 4.2 and 4.3 for positive XPath queries, further extending $\mathcal{X}(\downarrow, [\,], \neg)$ by allowing union ($\cup$) and upward modality ($\uparrow$) does not make the analysis harder (Theorem 5.2). The upper-bound proof involves a radically different techniques from those used in either the NP membership in the previous section, or the EXPTIME bounds of Neven and Schwentick [2006] and Marx [2004].

Proposition 5.1. $\mathsf{SAT}(\mathcal{X}(\downarrow, [\,], \neg))$ *is PSPACE-hard.*

Proof. We show that $\mathsf{SAT}(\mathcal{X}(\downarrow, [\,], \neg))$ is PSPACE-hard by a reduction from the Q3SAT problem, which is known to be PSPACE-complete (cf. Papadimitriou [1994]).

An instance of the Q3SAT problem is a well-formed quantified Boolean sentence $\phi = Q_1 x_1 Q_2 x_2 \cdots Q_m x_m E$, where $E = C_1 \wedge \cdots \wedge C_n$ is an instance of 3SAT in which all the propositional variables are $x_1, \ldots, x_m$, and $Q_i \in \{\forall, \exists\}$ for each $i \in [1, m]$. The Q3SAT problem is to decide, given such a quantified Boolean formula ϕ, whether or not ϕ is valid.

We next give a reduction from the Q3SAT problem to $\mathsf{SAT}(\mathcal{X}(\downarrow, [\,], \neg))$. Given a quantified Boolean formula ϕ as described above, we encode ϕ in terms of a DTD D and a query $\mathsf{XP}(\phi)$ in $\mathcal{X}(\downarrow, [\,], \neg)$. More specifically, we define the DTD

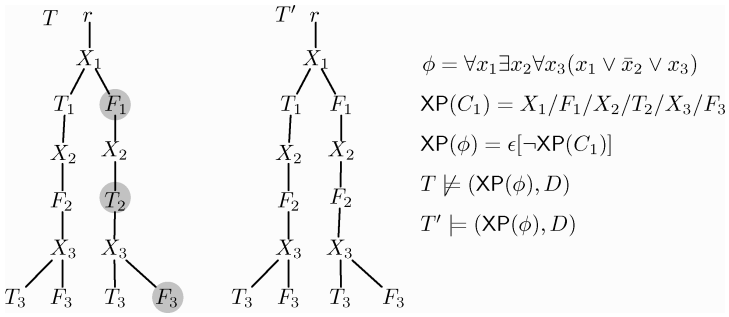

FIG. 3. Encoding of a Q3SAT instance in the proof of Proposition 5.1. The tree T does not satisfy $\mathsf{XP}(\phi)$ because of the truth assignment encoded in it indicated by the gray discs. On the other hand, T' satisfies $\mathsf{XP}(\phi)$. The truth assignments making ϕ true can be easily read from T'.

$D = (Ele, Att, P, R, r)$ as:

$Ele = \{X_j, T_j, F_j \mid j \in [1, m]\} \cup \{r\}$.
P: $r \to X_1$,
$X_i \to T_i \;\; op_i \;\; F_i$, /* for $i \in [1, m]$, where $op_i =$ ',' if $Q_i = \forall$,
and $op_i =$ '+' if $Q_i = \exists$ */
$T_i \to X_{i+1}$ /* for $i \in [1, m-1]$ */ $\quad F_i \to X_{i+1}$ /* for $i \in [1, m-1]$ */
$Att = \emptyset$, $R(A) = \emptyset$ for all $A \in Ele$.

As depicted as in Figure 3, an XML tree of D consist of multiple chains of $X_1, \ldots, X_m$, which encode propositional variables $x_1, \ldots, x_m$ in E, along with their possible truth assignments T_i, F_i for each X_i. More specifically, if the quantifier Q_i for x_i is $\forall$, then the production for X_i is defined in terms of a concatenation (T_i, F_i); intuitively, one needs to check whether ϕ holds under both T_i and F_i assignments for x_i; if Q_i is $\exists$, then the production for X_i is defined in terms of a disjunction $(T_i + F_i)$; in this case, one needs to check whether ϕ holds under either a T_i or F_i assignment for x_i. Observe that the size of an XML tree of D is possibly exponential in $|D|$, while $|D|$ is linear in the size of ϕ.

Furthermore, we encode ϕ in terms of a query $\mathsf{XP}(\phi) \in \mathcal{X}(\downarrow, [\,], \neg)$, defined with the following qualifiers at the root.

—*Encoding clauses*: For each clause C_i in ϕ we let $\mathsf{XP}(C_i)$ represent the *negation* of the clause C_i. More specifically, let $C_i = l_i^1 \vee l_i^2 \vee l_i^3$, where l_i^j is a literal, that is, it is either a variable x_i or the negation $\bar{x}_i$ of a variable. Then, $\mathsf{XP}(C_i)$ is to code $\neg l_i^1 \wedge \neg l_i^2 \wedge \neg l_i^3$. Without loss of generality, we may assume that the variables of these literals are x_s, x_t and x_u, respectively, with $s < t < u$. Then, $\mathsf{XP}(C_i)$ is defined as

$$\mathsf{XP}(C_i) = \downarrow^{(2s-2)}/X_s/Z_s/\downarrow^{(2(t-s)-2)}/X_t/Z_t/\downarrow^{(2(u-t)-2)}/X_u/Z_u,$$

where $Z_j = F_j$ if x_j appears in C_i, and $Z_j = T_j$ if $\bar{x}_j$ appears in C_i, for j ranging over s, t, u. Here $\downarrow^0 = \varepsilon$ and $\downarrow^j = \downarrow^{j-1}/\downarrow$.

—*Encoding ϕ*: We let $\mathsf{XP}(\phi)$ express that none of the negated clauses is satisfied for all truth assignments encoded in an instance conforming to D. In other words,

$$\mathsf{XP}(\phi) = \varepsilon[\neg \mathsf{XP}(C_1) \wedge \cdots \wedge \neg \mathsf{XP}(C_n)].$$

Figure 3 illustrates this encoding for $\phi = \forall x_1 \exists x_2 \forall x_3\ (x_1 \vee \bar{x}_2 \vee x_3)$.

We show that $(\mathsf{XP}(\phi), D)$ is satisfiable by an XML tree T if and only if the quantified Boolean formula ϕ is valid. Indeed, ϕ is satisfiable iff for any possible truth assignment for the universally quantified variables there exists a truth assignment for the remaining existentially quantified variables such that E evaluates to true. Each such truth assignment is encoded by a sequence of T_i's and F_i's along some branch in the tree T. Now, ϕ is satisfiable iff for each clause C_i in E, the truth assignment encoded by $\mathsf{XP}(C_i)$ does *not* appear on any branch in T (note that $\mathsf{XP}(C_i)$ encodes the negation of C_i), which in turn is true iff $\varepsilon[\neg\mathsf{XP}(C_i)]$ holds for each C_i. This verifies the correctness of the reduction. □

THEOREM 5.2. $\mathsf{SAT}(\mathcal{X})$ *is PSPACE-complete for any fragment* $\mathcal{X}$ *that contains* $\mathcal{X}(\downarrow, [\,], \neg)$*, and is contained in* $\mathcal{X}(\downarrow, \uparrow, \cup, [\,], \neg)$.

PROOF. By Proposition 5.1, it suffices to show that $\mathsf{SAT}(\mathcal{X}(\downarrow, \uparrow, \cup, [\,], \neg))$ is in PSPACE to obtain the PSPACE-completeness for the fragments specified in the theorem. The upper bound follows directly from the more general result that $\mathsf{SAT}(\mathcal{X}(\downarrow, \uparrow, \leftarrow, \rightarrow, \leftarrow^*, \rightarrow^*, \cup, [\,], \neg))$ is in PSPACE (Theorem 7.4). We refer for the proof of Theorem 7.4 to Section 7 where we consider XPath fragments with sibling axis . □

5.2. RECURSIVE XPATH QUERIES WITH NEGATIVE QUALIFIERS. We now study the impact of recursion $(\downarrow^*, \uparrow^*)$ on the satisfiability analysis. Recall from Theorem 4.4 that the presence of recursion does not make the analysis harder for positive XPath queries. In contrast, Theorem 7(ii) of Marx [2004] implies that the addition of recursion to $\mathcal{X}(\downarrow, [\,], \neg)$ makes the problem EXPTIME-hard. The upper bound in the same theorem implies that the bound above is tight, even when upward traversal $(\uparrow, \uparrow^*)$ and union $(\cup)$ are allowed (indeed, Marx [2004] shows that this upper bound holds even in the presence of specialized DTDs and sibling axes in queries).

THEOREM 5.3. [*Marx* 2004] $\mathsf{SAT}(\mathcal{X})$ *is EXPTIME-complete for any fragment* $\mathcal{X}$ *that* (1) *contains* $\mathcal{X}(\downarrow, \downarrow^*, [\,], \neg)$*, and* (2) *is contained in* $\mathcal{X}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \cup, [\,], \neg)$.

5.3. ADDING DATA VALUES. In contrast to Theorem 4.4, which shows that the presence of data values does not complicate the satisfiability analysis of positive XPath queries, we next show that adding data values to fragments with negation has an enormous impact on the analysis.

THEOREM 5.4. $\mathsf{SAT}(\mathcal{X}(\downarrow, \uparrow, \downarrow^*, \uparrow^*, \cup, [\,], =, \neg))$ *is undecidable*.

PROOF. We prove the undecidability by reduction from the halting problem for two-register machines, which is known to be undecidable (see, e.g., Börger et al. [1997]).

5.3.1. *Two Register Machine*. A two-register machine (2RM) M has two registers $\mathsf{register}_1$, $\mathsf{register}_2$, and is programmed by a numbered sequence $I_0, I_1, \ldots, I_l$ of instructions. Each register contains a natural number. An *instantaneous description* (*ID*) of M is (i, m, n), where $i \in [0, l]$, m and n are natural numbers. It indicates that M is to execute instruction I_i (or at "state i") with $\mathsf{register}_1$ and $\mathsf{register}_2$ containing m and n, respectively.

An instruction I_i of M can be either an addition or a subtraction, which defines a relation $\rightarrow_M$ between IDs, described as follows:

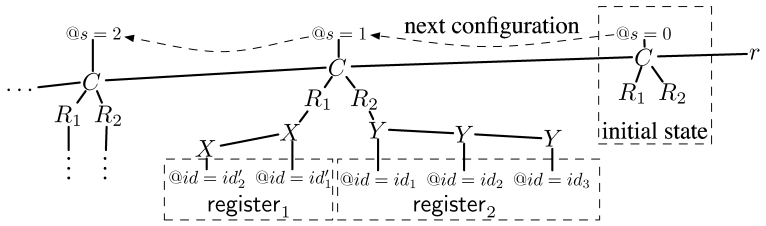

FIG. 4. An XML tree of the DTD encoding a 2RM in the proof of Theorem 5.4

—*addition*: (i, rg, j), where rg is either $\mathsf{register}_1$ or $\mathsf{register}_2$, and $0 \leq i, j \leq l$. Its semantics is: at state i, M adds 1 to the content of rg, and then goes to state j. Accordingly:

$$(i, m, n) \rightarrow_M \begin{cases} (j, m+1, n) & \text{if } \mathsf{rg} = \mathsf{register}_1 \\ (j, m, n+1) & \text{otherwise} \end{cases}$$

—*subtraction*: (i, rg, j, k), where rg is either $\mathsf{register}_1$ or $\mathsf{register}_2$, and $0 \leq i, j, k \leq l$. Its semantics is: at state i, M tests whether the content of rg is 0, and if it is, then goes to state j; otherwise M subtracts 1 from rg and goes to the state k. Accordingly:

$$(i, m, n) \rightarrow_M \begin{cases} (j, 0, n) & \text{if } \mathsf{rg} = \mathsf{register}_1 \text{and } m = 0 \\ (k, m-1, n) & \text{if } \mathsf{rg} = \mathsf{register}_1 \text{ and } m \neq 0 \\ (j, m, 0) & \text{if } \mathsf{rg} = \mathsf{register}_2 \text{ and } n = 0 \\ (k, m, n-1) & \text{if } \mathsf{rg} = \mathsf{register}_2 \text{ and } n \neq 0. \end{cases}$$

Assume, without loss of generality, that the initial ID is $I = (0, 0, 0)$ and that the final ID is $F = (f, 0, 0)$, that is, a halting state $f \in [0, l]$ with zeros in both registers. The *halting problem for 2RM* is to determine, given a 2RM M as described above, whether or not $I \Rightarrow_M F$, where $\Rightarrow_M$ denotes the reflexive and transitive closure of $\rightarrow_M$.

5.3.2. *Reduction.* We now provide a reduction from the halting problem for 2RM to $\mathsf{SAT}(\mathcal{X}(\downarrow, \uparrow, \downarrow^*, \uparrow^*, \cup, [\ ], =, \neg))$. Given an instance M of 2RM as described above, we encode M in terms of a DTD D and an XPath query $p \in \mathcal{X}(\downarrow, \uparrow, \downarrow^*, \uparrow^*, \cup, [\ ], =, \neg)$.

More specifically, we define the DTD $D = (Ele, Att, P, R, r)$ as follows:

$$\begin{array}{ll} Ele = \{r,\ C,\ R_1,\ R_2,\ X,\ Y\}. & \\ P: \quad r \rightarrow C, & C \rightarrow (C, R_1, R_2) + \varepsilon, \quad R_1 \rightarrow X + \varepsilon \quad R_2 \rightarrow Y + \varepsilon, \\ \quad X \rightarrow X + \varepsilon, & Y \rightarrow Y + \varepsilon. \\ Att = \{@s,\ @id\}, & R(C) = \{@s\}, \qquad R(X) = R(Y) = \{@id\}. \end{array}$$

As depicted in Figure 4, an XML tree of the DTD D consists of an unbounded chain of C elements, coding the execution of the 2RM M starting from the first C element on the chain. Each C element encodes an ID of M: it has an s attribute indicating the state of the ID, and two children R_1, R_2 coding the contents of $\mathsf{register}_1$ and $\mathsf{register}_2$ of the ID, respectively. More specifically, R_1 has a chain of X elements, and the length of the X-chain encodes the content of $\mathsf{register}_1$; to count the number of X elements in the chain, an $X.id$ attribute is defined for X elements, which is to serve as a *local key* for the X elements on the chain; similarly for R_2 and the chain of Y elements under R_2.

We also use the following qualifiers at the root to encode the 2RM M, expressed in $\mathcal{X}(\downarrow, \uparrow, \downarrow^*, \uparrow^*, \cup, [\,], =, \neg)$.

(1) *Initial ID*. We code the initial ID $(0, 0, 0)$ of M by using the first C element on the C-chain under the root r.

$$Q_{start} = C[(\varepsilon/@s = 0 \ \wedge \ R_1[\neg X] \ \wedge \ R_2[\neg Y])].$$

(2) *Halting state*. The final ID $(f, 0, 0)$ of M is expressed as

$$Q_{halting} = \downarrow^*/C[\varepsilon/@s = f \ \wedge \ R_1[\neg X] \ \wedge \ R_2[\neg Y]].$$

This asserts that the ID $(f, 0, 0)$ (i.e., a C element on the C chain under r) can be reached.

(3) *Local key*. To count the number of X (respectively, Y) elements in an X-chain (resp. Y-chain) under a C element, we enforce $X.id$ to be a key for each X-chain (respectively, Y-chain):

$$Q_{xKey} = \neg\downarrow^*/X[\varepsilon/@id = \downarrow/\downarrow^*/@id], \quad Q_{yKey} = \neg\downarrow^*/Y[\varepsilon/@id = \downarrow/\downarrow^*/@id].$$

This suffices as it asserts that the $X.id$ of any X element is different from the $X.id$ of any of its X descendant (on the same chain); in other words, all the X elements on a chain under R_1 have distinct $X.id$ values; similarly for Y elements.

(4) *Transition*. For each $i \in [0, l]$, we code the ith instruction I_i with a qualifier Q_i, based on the type of I_i.

(Case 1: *Addition*). If I_i is an addition (i, rg, j), where $\mathsf{rg} = \mathsf{register}_1$, then Q_i is defined to ensure that *for any* C element c_1 with state $c_1.s = i$, (i) the next C element c_2 on the C-chain has state $c_2.s = j$ (state change); (ii) the X-chain of c_2 has one more element than that of c_1 ($\mathsf{register}_1$ is incremented by 1); and moreover, (iii) the length of the Y-chain of c_2 is the same as that of c_1 ($\mathsf{register}_2$ remains unchanged). These are expressed as:

$$\begin{aligned}
Q_i &= \neg\downarrow^*/C[\varepsilon/@s = i \ \wedge (C/@s \neq j \ \vee \ Q_a^X \ \vee \ Q^Y)] \\
Q_a^X &= R_1/\downarrow/\downarrow^*[\neg(\varepsilon/@id = \uparrow^*[\mathsf{lab}() = R_1]/\uparrow/C/R_1/\downarrow/\downarrow^*[X]/@id)] \\
&\quad \vee C/R_1/\downarrow/\downarrow^*[X \ \wedge \ \neg(\varepsilon/@id = \uparrow^*[\mathsf{lab}() = R_1]/\uparrow/\uparrow/R_1/\downarrow/\downarrow^*/@id)] \\
Q^Y &= R_2/\downarrow/\downarrow^*[\neg(\varepsilon/@id = \uparrow^*[\mathsf{lab}() = R_2]/\uparrow/C/R_2/\downarrow/\downarrow^*/@id)] \\
&\quad \vee C/R_2/\downarrow/\downarrow^*[\neg(\varepsilon/@id = \uparrow^*[\mathsf{lab}() = R_2]/\uparrow/\uparrow/R_2/\downarrow/\downarrow^*/@id)].
\end{aligned}$$

Here, Q_i, Q_a^X and Q^Y assert the conditions (i – iii) above, capitalizing on the local key $@id$ for each X (respectively, Y) chain. Similarly, Q_i, Q_a^X and Q^Y can be defined for $\mathsf{rg} = \mathsf{register}_2$.

(Case 2: *Subtraction*). If I_i is a subtraction (i, rg, j, k), where $\mathsf{rg} = \mathsf{register}_1$, then Q_i is defined to ensure that *for any* C element c_1 with state $c_1.s = i$, (i) the next C element c_2 on the C-chain has state $c_2.s = j$ if c_1 has no X subelement (i.e., $\mathsf{register}_1$ is 0), and furthermore, c_2 has no X subelement and it has the same number of Y subelements as c_1; in other words, the contents of both registers remain unchanged; and (ii) if c_1 has an X subelement (i.e., $\mathsf{register}_1 \neq 0$), then c_2 has state $c_2.s = k$, and moreover, the X-chain of c_2 has one less element than that of c_1 ($\mathsf{register}_1$ is decremented by 1), while the length of the Y-chain of c_2 is the same

as that of c_1 ($\mathsf{register}_2$ remains unchanged). These are expressed as follows:

$$
\begin{aligned}
Q_i &= \neg\downarrow^*/C[\varepsilon/@s = i \ \wedge (Q_s^0 \ \vee \ Q_s^X)] \\
Q_s^0 &= (R_1[\neg X] \ \wedge \ (C/@s \neq j \ \vee \ C/R_1[X] \ \vee \ Q^Y)) \\
Q_s^X &= (R_1[X] \ \wedge \ (C/@s \neq k \\
&\qquad \vee R_1/\downarrow/\downarrow^*[X \ \wedge \ \neg(\varepsilon/@id = \uparrow^*[\mathsf{lab}() = R_1]/\uparrow/C/R_1/\downarrow/\downarrow^*/@id)] \\
&\qquad \vee C/R_1/\downarrow/\downarrow^*[\neg(\varepsilon/@id = \uparrow^*[\mathsf{lab}() = R_1]/\uparrow/\uparrow/R_1/\downarrow/\downarrow^*[X]/@id)]))
\end{aligned}
$$

Here Q^Y is the same as defined in Case 1, and Q_i, Q_s^0 and Q_s^X assert the conditions (i–ii) above. Similarly, Q_i, Q_s^0 and Q_s^X can be defined for $\mathsf{rg} = \mathsf{register}_2$. Putting these together, we define the query p to be

$$\varepsilon[Q_{start} \ \wedge \ Q_{halting} \ \wedge \ Q_{xKey} \ \wedge \ Q_{yKey} \ \wedge \ \bigwedge_{i\in[0,l]} Q_i].$$

One can verify that p is in $\mathcal{X}(\downarrow, \uparrow, \downarrow^*, \uparrow^*, \cup, [\ ], =, \neg)$ and furthermore, that (p, D) is satisfiable iff the 2RM M halts, that is, $(0, 0, 0) \Rightarrow_M (f, 0, 0)$. □

The good news is that not every fragment with negation and data values is beyond reach: below we show that the satisfiability problem in the presence of data values and negation is still decidable for nonrecursive downward queries. Unlike the previous results, the proof uses a finite-model-theoretic construction. This and Theorem 5.4 indicate that XPath with negation and data values is very close to the border of decidability, and which side a particular fragment falls on depends on syntactic restrictions on axes and qualifier constructs.

THEOREM 5.5. $\mathsf{SAT}(\mathcal{X}(\downarrow, \cup, [\ ], =, \neg))$ *is in NEXPTIME.*

PROOF. We show that there is a NEXPTIME algorithm that, given a query p in $\mathcal{X}(\downarrow, \cup, [\ ], =, \neg)$ and a DTD D, determines whether or not (p, D) is satisfiable. It suffices to show that if (p, D) has a model (i.e., there exists an XML tree satisfying (p, D)), then it has a model of exponential size in $|p|$ and $|D|$. For if this holds, then a nondeterministic decision algorithm works as follows: first guess a model T of size exponential in $|p|$ and $|D|$, and then check whether T satisfies (p, D). The latter can be done in polynomial time in $|T|$ and $|p|$ (cf. Gottlob et al. [2005]), and thus the algorithm is in NEXPTIME. Note that it is not necessary to guess and check all possible data values for attributes. The decision algorithm only needs to guess a binary relation '=' between the attribute values and between attribute values and constants mentioned in p. Such a relation has a size bounded by $O(N(T)|p|^2)$, where $N(T)$ is the number of nodes in a satisfying tree. Hence, it suffices to show that the number of nodes in a satisfying model can be taken to be exponential in p.

Our next goal is to show this "small model property". More specifically, we show that if (p, D) has a model, then it has a model where the underlying tree structure has depth bounded by $|p|$ and the out-degree of each node is bounded by $|D| + |p|$. Observe that $|p|$ is a bound on the "depth" of any XML tree T satisfying p – that is, to determine whether or not T satisfies p, one only needs to look at the subtree of T down to the level $|p|$ from the root, because p does not have the recursive axes. As a result, the decision algorithm only needs to guess a tree up to depth $|p|$. We only need to show that the width of the tree (i.e., the maximum number of children under each node) is bounded by $l = |D| + |p|$, referred to as the *bounded width property*. For if this holds, then the size of the tree is at most $(2^l)^{|p|} = 2^{(l\times|p|)}$, exponential in the size of the input query p and DTD D.

Suppose that there is an XML tree T satisfying both p and the DTD D. We find from T a small submodel T_1 with the bounded-width property. That is, we show that T has a "small" (linear depth and linear width, hence exponential size) model T_1 of p embedded in it. To derive T_1, we start with a substructure T_0 of T such that (1) the nodes in T_0 are closed under ancestor; (2) T_0 has the structure and attributes inherited from T; (3) T_0 satisfies D; (4) every node in T_0 has at most $|D|$ children in T_0. This is possible since $T \models D$.

We now give an algorithm $\mathsf{witness}(n, T_0)$ which will return for each node n a set of nodes in T. We will use this algorithm to extend T_0 by adding nodes. The algorithm $\mathsf{witness}(n, T_0)$ does the following. For every positive subquery p' of p that is satisfied at n in T, $\mathsf{witness}(n, T_0)$ includes witnesses that will guarantee p' is satisfied by n in T_0: in particular, for a qualifier of the form $p_1/@a \ \mathsf{op} \ p_2/@b$ in p that is satisfied by n in T, we choose witnesses $n_1 \in n[\![p_1]\!]$, $n_2 \in n[\![p_2]\!]$ such that $n_1.a \ \mathsf{op} \ n_2.b$, and put n_1, n_2 and all of their ancestors that are also descendants of n (including n itself) into $\mathsf{witness}(n, T_0)$. For each qualifier of the form p_1 satisfied at n, we throw in $n_1 \in n[\![p_1]\!]$ and the ancestors of n_1 that are descendants of n into $\mathsf{witness}(n, T_0)$. Finally, we recursively call $\mathsf{witness}(n', T_0)$ on all children n' of n in T that are either in $\mathsf{witness}(n, T_0)$ or in T_0. From $\mathsf{witness}(n, T_0)$, we construct an XML tree that consist of all the top-level witnesses and the union of the witnesses returned by the recursive calls. Note that the top-level call to $\mathsf{witness}(n, T_0)$ adds at most $|p|$ children of n into the output set, since every witness of a positive qualifier generates at most one child of n in the output. Since the top-level call returns only n and descendants of n into the output, the recursive calls on children of n will not themselves add any new children of n to the output. Hence, $\mathsf{witness}(n, T_0)$ returns a set of nodes of T that includes at most $|p|$ children of n.

Let T_1 be the result of adding to T_0 all nodes in $\mathsf{witness}(root(T), T_0)$. Note that T_1 has the tree and attribute structure inherited from T. Since T_0 satisfies D, T satisfies D, and T_1 was obtained by expanding T_0 within T, T_1 still satisfies D (recall that D is assumed to be normalized). Since $\mathsf{witness}(n, T_0)$ is closed under ancestor, the result is a valid XML tree, and by the observation above about $\mathsf{witness}(n, T_0)$ the width of T_1 is at most $|D| + |p|$. Note that T_1 has the property that if $n \in T_1$, $\mathsf{witness}(n, T_0)$ is a subset of T_1.

We now show that for every $n \in T_1$, for every qualifier q that is a subexpression of p, n satisfies q in T_1 iff n satisfies q in T. The proof is by induction on q. Since this is an "iff", the negation step is immediate, as are the other Boolean connectives. Suppose q is of the form $p_1/@a \ \mathsf{op} \ p_2/@b$. If $n \in T_1$ satisfies q in T, then there are $n_1, n_2 \in T_1$ such that $T \models n_1 \in n[\![p_1]\!]$, $T \models n_2 \in n[\![p_2]\!]$, and $n_1.a \ \mathsf{op} \ n_2.b$. But by induction, $T_1 \models n_1 \in n[\![p_1]\!]$, $T_1 \models n_2 \in n[\![p_2]\!]$, and hence n satisfies q in T_1. If n satisfies q in T_1, then there are $n_1, n_2 \in T_1$ such that $T_1 \models n_1 \in n[\![p_1]\!]$, $T_1 \models n_2 \in n[\![p_2]\!]$, and $n_1.a \ \mathsf{op} \ n_2.b$. But since T_1 is a substructure of T we have $n_1, n_2 \in T$, and we have $T \models n_1 \in n[\![p_1]\!]$ and $T \models n_2 \in n[\![p_2]\!]$ by induction. Hence n satisfies q in T. The case of qualifiers of the form p_1 is similar. Applying this equivalence to a top-level filter and n the root of T, we see that $(T, root(T)) \models p$ iff $(T_1, root(T)) \models p$. □

While the decidability of the satisfiability problem remains open if the upward axis $\uparrow$ is further added, we next show that the "hardness" increases compared to the fragment without data equality. Contrast the result below with Proposition 5.1.

THEOREM 5.6. $\mathsf{SAT}(\mathcal{X}(\uparrow, [\ ], =, \neg))$ *is EXPTIME-hard,*

PROOF. We prove this by reduction from the two-player game of corridor tiling (TPG-CT), which is EXPTIME-complete [Chlebus 1986]. Our reduction below was inspired by Neven and Schwentick [2006], which establishes the lower bound for the containment problem for $\mathcal{X}(\downarrow, \downarrow^*, [\,])$. The TPG-CT problem can be stated as follows.

5.3.3. *TPG-CT.* An instance of TPG-CT consists of a tiling system $(X, H, V, \vec{t}, \vec{b})$ and a natural number n, where X is a finite set of dominoes (tiles), $H, V \subseteq X \times X$ are two binary relations, $\vec{t}$ and $\vec{b}$ are two n-vectors of given tiles in X, and n is the number of columns (the width of the corridor). It is to determine whether or not player I has a winning strategy for tiling the corridor. By tiling the corridor we mean that there exists a tiling $\tau : \mathbb{N} \times \mathbb{N} \to X$ and a natural number m such that for all $x \in [1, n]$ and $y \in [1, m]$, the *tiling adjacency conditions* are observed, that is,

—if $\tau(x, y) = d$ and $\tau(x + 1, y) = d'$, then $(d, d') \in H$;
—if $\tau(x, y) = d$ and $\tau(x, y + 1) = d'$, then $(d, d') \in V$;
—$\tau(x, 1) = \vec{t}[x]$ and $\tau(x, m) = \vec{b}[x]$, where $\vec{t}[x]$ (respectively, $\vec{b}[x]$) denotes the xth element of the vector $\vec{t}$ (respectively, $\vec{b}$); that is, the given tiles of $\vec{t}$ and $\vec{b}$ are placed in the top and bottom rows, respectively.

Player I and Player II in turn place a tile from X in the first free location (row by row from left to right), observing the tiling adjacency conditions. The one who is unable to make the next move loses the game. Player II may also decide to end the game and check the tiling adjacency conditions at the bottom row. The given tiles $\vec{t}$ and $\vec{b}$ are placed in the top and bottom rows by the referee of the game. The problem is EXPTIME-complete even when n is even [Chlebus 1986], and thus below we assume that n is even.

5.3.4. *Reduction.* Given an instance, $(X, H, V, \vec{t}, \vec{b})$ and n, of TPG-CT, we define a DTD D_0 and a query Q in $\mathcal{X}(\uparrow, [\,], =, \neg)$ such that there exists an XML tree T satisfying (Q, D_0) if and only if Player I has a winning strategy for the game. Let $X = \{x_1, \ldots, x_k\}$. To simplify the presentation, we use disjunction in the coding query Q; this does not lose generality since disjunction can be easily eliminated by using negation and conjunction. We assume without loss of generality that Player I moves first.

We define a nonrecursive and disjunction-free DTD $D_0 = (Ele, Att, P, R, r)$, where

$$Ele = \{r, C\}.$$
$$P: \quad r \to C^*, \qquad C \to \varepsilon.$$
$$Att = \{@h, @k, @\mathsf{next}\} \cup \{@t_i \mid i \in [1, n]\}, \qquad R: R(C) = Att,\ R(r) = \emptyset.$$

An XML tree of D_0 consists merely of a root node and its C children. Each C element v codes a snapshot of the game showing the last n plays, where n is the length of a row in the corridor. The attributes are used as follows: (1) $v.h$ codes the horizontal position of the last tile $v.t_n$ in a row, (2) $v.t_i$ represents a tile in X placed by a player, (3) $v.k$ and $v.\mathsf{next}$ encode a list of such snapshots: $v.k$ is the "identifier" of the current snapshot, while $v.\mathsf{next}$ is a pointer to the next one. (See Figure 5.)

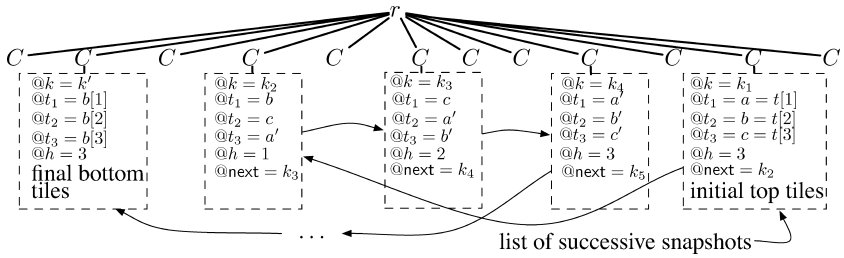

FIG. 5. Encoding of a TPG-CT instance in the proof of Theorem 5.6, with $n = 3$ (the number of tiles in a row).

We use qualifiers in $\mathcal{X}(\uparrow, \cup, [\,], =, \neg)$ to encode the following:
The Ranges of Attribute Values. The range of attribute h is $[1, n]$, and the range of attribute t_i is X. This is expressed as a qualifier $Q_{(h,t)}$ as follows:

$$Q_{(h,t)} = \neg\left(C\left[\bigwedge_{i\in[1,n]}(\varepsilon/@h \neq i) \;\vee\; \bigvee_{i\in[1,n]}\bigwedge_{j\in[1,k]}(\varepsilon/@t_i \neq x_j)\right]\right).$$

Key. For any nodes v and v', if they have the same value for attribute k, then they must have the same values for attributes t_i and h. We state this as a qualifier Q_u as follows.

$$Q_u = \neg\left(C\left[\bigvee_{i\in[1,n]}(\varepsilon/@h = i \;\wedge\; \varepsilon/@k = \uparrow/C[\varepsilon/@h \neq i]/@k)\right.\right.$$
$$\left.\left.\vee \bigvee_{i\in[1,n]}\bigvee_{j\in[1,k]}(\varepsilon/@t_i = x_j \;\wedge\; \varepsilon/@k = \uparrow/C[\varepsilon/@t_i \neq x_j]/@k)\right]\right).$$

Consistency of Successor. For any nodes v and v', if $v.\mathsf{next} = v'.k$, then they must satisfy the following: (1) if $v.h = n$, then $v'.h = 1$; (2) if $v.h = l < n$, then $v'.h = l + 1$; and (3) $v.t_i = v'.t_{i-1}$ for $2 \le i \le n$. We state this as a qualifier Q_s as follows.

$$Q_s = \neg\Big(C\Big[(\varepsilon/@h = n \;\wedge\; \varepsilon/@\mathsf{next} = \uparrow/C[\varepsilon/@h \neq 1]/@k)$$
$$\vee \bigvee_{i\in[1,n-1]}(\varepsilon/@h = i \;\wedge\; \varepsilon/@\mathsf{next} = \uparrow/C[\varepsilon/@h \neq i+1]/@k)$$
$$\vee \bigvee_{i\in[2,n]}\bigvee_{j\in[1,k]}(\varepsilon/@t_i = x_j \;\wedge\; \varepsilon/@\mathsf{next} = \uparrow/C[\varepsilon/@t_{i-1} \neq x_j]/@k)\Big]\Big).$$

Initial Values. There is a C node v with $v.h = n$ such that its attributes $v.t_1, \ldots, v.t_n$ match the initial top tiles $\vec{t}$. This is stated as follows:

$$Q_0 = C\left[\bigwedge_{i\in[1,n]}(\varepsilon/@t_i = \vec{t}[i])\right].$$

Adjacency Constraints. For any nodes v and v', if $v.\mathsf{next} = v'.k$, then they must satisfy the vertical tiling constraint: $(v.t_1, v'.t_n) \in V$. Moreover, any node v must satisfy the horizontal constraint: $(v.t_i, v.t_{i+1}) \in H$ for all $i \in [1, n-1]$. We express this as Q_c:

$$Q_c \;=\; \neg C\left[\bigwedge_{(x,x')\in V}(\varepsilon/@t_1 = x \;\wedge\; \varepsilon/@\mathsf{next} = \uparrow/C[\varepsilon/@t_n \neq x']/@k) \;\vee\; \left(\bigvee_{i\in[1,n-1]}\;\bigwedge_{(x,x')\in H}(\varepsilon/@t_i = x \;\wedge\; \varepsilon/@t_{i_1} \neq x')\right)\right].$$

Play Continues unless Player I Has Won. For any node v, if $v.h < n$, then there is some v' with $v'.k = v.\mathsf{next}$. If $v.h = n$ and if the bottom vector $\vec{b}$ is not matched, that is, for some $i \in [1, n]$, $v.t_i \neq \vec{b}[i]$, then there is some v' with $v'.k = v.\mathsf{next}$. We express this as Q_p:

$$Q_p \;=\; \neg C\left[\bigvee_{i\in[1,n-1]}(\varepsilon/@h = i \;\wedge\; \neg(\varepsilon/@\mathsf{next} = \uparrow/C/@k)) \;\vee\left(\varepsilon/@h = n \wedge \bigvee_{i\in[1,n-1]}(\varepsilon/@t_i \neq \vec{b}[i]) \wedge \neg(\varepsilon/@\mathsf{next} = \uparrow/C/@k)\right)\right].$$

Player I Has to Respond to All Possible Moves of Player II. For any node v, if $v.h$ is odd, that is, the last move $v.t_n$ was made by Player I, then for any tile $x \in X$ such that it satisfies the horizontal constraint $(v.t_n, x) \in H$ and the vertical constraint $(v.t_1, x) \in V$, there is some node v' with $v.\mathsf{next} = v'.k$ and $v'.t_1 = x$. That is, all possible moves of Player II have to be considered. We encode this in terms of a qualifier $Q_\forall$. Let N_{odd} be the set of odd natural numbers in $[1, n]$.

$$Q_\forall \;=\; \neg C\left[\bigvee_{i\in N_{odd}}(\varepsilon/@h = i \;\wedge\; \bigvee_{j\in[1,k]}\left(\bigvee_{(x,x_j)\in H}\varepsilon/@t_n = x\right) \;\wedge\; \left(\bigvee_{(x,x_j)\in V}\varepsilon/@t_1 = x\right) \;\wedge\; \neg(\varepsilon/@\mathsf{next} = \uparrow/C[\varepsilon/@t_1 = x_j]/@k))\right].$$

Putting these together, the coding of the TPG-CT instance consists of the DTD D_0 and the $\mathcal{X}(\uparrow, \cup, [\,], =, \neg)$-query $Q = \varepsilon[Q_{(h,t)} \wedge Q_u \wedge Q_s \wedge Q_0 \wedge Q_c \wedge Q_p \wedge Q_\forall]$.

We now verify that Player I has a winning strategy if and only there is an XML tree T such that $T \models (Q, D_0)$. First, suppose that T is an XML tree satisfying the above. We give a wining strategy for Player I. Player I begins with the node given by the initial value qualifier Q_0. At any point i in the game, Player I has a node v_i in T that is inductively assumed to represent the last n moves of the play thus far. If Player II plays a tile x_j as the next move, the qualifier $Q_\forall$ gets a node v' with $v'.k = v_i.\mathsf{next}$ and $v'.t_1 = x_j$. By the qualifiers Q_u, Q_s and Q_c, v' satisfies the horizontal and vertical constraints, and it also correctly updates the last n tiles played. Then, the qualifier Q_p gets a node v'', which corresponds to a move by Player I. Again by the qualifiers Q_u, Q_c, Q_s, v'' is a valid response. Conversely, if Player I has a winning strategy, we can form an XML tree whose nodes consist of all valid plays in any game, where each node codes the horizontal position of the last move in a row and the last n tiles played in the game. It is easy to confirm that this satisfies the query Q above and the XML tree conforms to the DTD D_0. □

5.4. CONTAINMENT ANALYSIS IN THE PRESENCE OF NEGATION. The results above, along with Proposition 3.2, give us complexity bounds for the containment problem for XPath fragments with negation in the presence of DTDs.

COROLLARY 5.7. *For the containment problem* $\mathsf{CNT}(\mathcal{X})$ *in the presence of DTDs,*

(1) $\mathsf{CNT}(\mathcal{X}(\downarrow, [\,], \neg))$ *is PSPACE-hard;*
(2) $\mathsf{CNT}(\mathcal{X}(\downarrow, \uparrow, \cup, [\,], \neg))$ *is PSPACE-complete;*
(3) $\mathsf{CNT}(\mathcal{X}(\downarrow, \downarrow^*, [\,], \neg))$ *is EXPTIME-hard;*
(4) $\mathsf{CNT}(\mathcal{X}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \cup, [\,], \neg))$ *is EXPTIME-complete;*
(5) $\mathsf{CNT}(\mathcal{X}(\downarrow, \uparrow, \cup, [\,], =, \neg))$ *is EXPTIME-hard;*
(6) $\mathsf{CNT}(\mathcal{X}(\downarrow, \cup, [\,], =, \neg))$ *is in coNEXPTIME;*
(7) $\mathsf{CNT}(\mathcal{X}(\downarrow, \uparrow, \downarrow^*, \uparrow^*, \cup, [\,], =, \neg))$ *is undecidable.*

6. *Satisfiability Analysis under Restricted DTDs*

The hardness results in the previous section leave open the possibility that feasible algorithms exist for restricted DTDs that may occur often in practice. We thus investigate whether or not restricted DTDs simplify the analysis of XPath satisfiability. More specifically, we study $\mathsf{SAT}(\mathcal{X})$ for a variety of XPath fragments $\mathcal{X}$ in the following four settings: (1) when DTDs are non-recursive; (2) when DTDs are fixed; (3) when DTDs are disjunction-free; and (4) in the absence of DTDs, which, as shown by Proposition 3.1, is reducible to a special case of $\mathsf{SAT}(\mathcal{X})$. We show that for some restricted DTDs and some fragments $\mathcal{X}$, $\mathsf{SAT}(\mathcal{X})$ has lower complexity. Note that the upper bounds for $\mathsf{SAT}(\mathcal{X})$ also hold for the restricted analysis: if $\mathsf{SAT}(\mathcal{X})$ is in a complexity class K, then the satisfiability analysis is also in K under restricted DTDs.

6.1. NONRECURSIVE DTDS. A nonrecursive DTD D has the property that for any XML tree T conforming to D, the depth of T, that is, the length of the longest path from the root to a leaf of T, is bounded by $|D|$. This simplifies the analysis of XPath queries with recursive axes ($\downarrow^*$, $\uparrow^*$). Specifically, for any $\mathcal{X}(\downarrow, \downarrow^*, \cup, \dots)$, that is, a fragment with $\downarrow$, recursion $\downarrow^*$, union $\cup$ and possibly other operators, $\mathsf{SAT}(\mathcal{X}(\downarrow, \downarrow^*, \cup, \dots))$ and $\mathsf{SAT}(\mathcal{X}(\downarrow, \cup, \dots))$ are cubic-time equivalent under nonrecursive DTDs, where $\mathcal{X}(\downarrow, \cup, \dots)$ denotes the same fragment without $\downarrow^*$. Indeed, there is a cubic-time reduction from $\mathsf{SAT}(\mathcal{X}(\downarrow, \downarrow^*, \cup, \dots))$ to $\mathsf{SAT}(\mathcal{X}(\downarrow, \cup, \dots))$. Intuitively, given a nonrecursive DTD D, one can eliminate recursion in a query by replacing $\downarrow^*$ with $(\varepsilon \cup \downarrow \cup \cdots \cup \downarrow^{|D|})$, where $\downarrow^n$ abbreviates the n-fold concatenation of $\downarrow$; similarly for $\mathcal{X}(\uparrow, \uparrow^*, \cup, \dots)$, that is, a fragment with $\uparrow$, $\uparrow^*$, $\cup$ and other operators. This is stated below.

PROPOSITION 6.1. *Under nonrecursive DTDs, the following problems are cubic-time equivalent:*

(1) $\mathsf{SAT}(\mathcal{X}(\downarrow, \downarrow^*, \cup, \dots))$, $\mathsf{SAT}(\mathcal{X}(\downarrow, \cup, \dots))$;
(2) $\mathsf{SAT}(\mathcal{X}(\uparrow, \uparrow^*, \cup, \dots))$, $\mathsf{SAT}(\mathcal{X}(\uparrow, \cup, \dots))$;
(3) $\mathsf{SAT}(\mathcal{X}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \cup, \dots))$, $\mathsf{SAT}(\mathcal{X}(\downarrow, \uparrow, \cup, \dots))$;

where $\mathcal{X}(\downarrow, \uparrow, \cup, \dots)$ *supports the same set of operators as* $\mathcal{X}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \cup, \dots)$ *except* $\downarrow^*$, $\uparrow^*$; *similarly for* $\mathcal{X}(\downarrow, \cup, \dots)$ *and* $\mathcal{X}(\uparrow, \cup, \dots)$.

PROOF. We prove that under non-recursive DTDs, $\mathsf{SAT}(\mathcal{X}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \cup, \dots))$ and $\mathsf{SAT}(\mathcal{X}(\downarrow, \uparrow, \cup, \dots))$ are cubic-time equivalent; the other statements of the proposition can be verified similarly.

Given any nonrecursive DTD D and any query $p \in \mathcal{X}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \cup, \dots)$, we rewrite p to $p' \in \mathcal{X}(\downarrow, \uparrow, \cup, \dots)$ by replacing each occurrence of $\downarrow^*$ in p with $(\varepsilon \cup \downarrow \cup \cdots \cup \downarrow^{|D|})$, and each occurrence of $\uparrow^*$ with $(\varepsilon \cup \uparrow \cup \ldots \cup \uparrow^{|D|})$. One can easily verify that over any XML tree T of D, p and p' are equivalent, i.e., $r[\![p]\!] = r[\![p']\!]$, where r is the root of T. Indeed, since the depth of T is bounded by $|D|$, the equivalence can be verified by a straightforward induction on the structure of p. Moreover, the rewriting takes at most $O(|p||D|^2)$ time. □

From the proposition it follows that the EXPTIME problem of Theorem 5.3 collapses to PSPACE (Theorem 5.2), and that $\mathsf{SAT}(\mathcal{X}(\downarrow, \downarrow^*, \cup, [\,], =, \neg))$ is now known to be decidable (Theorem 5.5) under nonrecursive DTDs.

COROLLARY 6.2. *Under nonrecursive DTDs,*

(1) $\mathsf{SAT}(\mathcal{X}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \cup, [\,], \neg))$ *is in PSPACE, and*
(2) $\mathsf{SAT}(\mathcal{X}(\downarrow, \downarrow^*, \cup, [\,], =, \neg))$ *is in NEXPTIME.*

They are equivalent to $\mathsf{SAT}(\mathcal{X}(\downarrow, \uparrow, \cup, [\,], \neg))$ *and* $\mathsf{SAT}(\mathcal{X}(\downarrow, \cup, [\,], =, \neg))$*, respectively.*

One might be tempted to think that nonrecursive DTDs might also lower the NP, PSPACE and EXPTIME lower bounds given earlier. However, the proofs of Propositions 4.2, 4.3, Theorem 5.2 and Theorem 5.6 all utilize nonrecursive DTDs. Hence:

COROLLARY 6.3. *Under nonrecursive DTDs,*

(1) $\mathsf{SAT}(\mathcal{X}(\downarrow, [\,]))$, $\mathsf{SAT}(\mathcal{X}(\cup, [\,]))$ *and* $\mathsf{SAT}(\mathcal{X}(\downarrow, \uparrow))$ *are NP-hard;*
(2) $\mathsf{SAT}(\mathcal{X}(\downarrow, [\,], \neg))$ *is PSPACE-hard.*
(3) $\mathsf{SAT}(\mathcal{X}(\uparrow, \cup, [\,], =, \neg))$ *is EXPTIME-hard.*

6.2. FIXED DTDS. Under fixed DTDs, the satisfiability problem $\mathsf{SAT}(\mathcal{X})$ is to determine, given any query $p \in \mathcal{X}$, whether or not there exists an XML tree T that satisfies both p and a fixed DTD D_0. Here, D_0 is not an input, but is predefined. Together with other restrictions, fixed DTDs may simplify the analysis of $\mathsf{SAT}(\mathcal{X})$. For example, the observation below contrasts with the EXPTIME hardness in Theorem 5.3.

PROPOSITION 6.4. $\mathsf{SAT}(X(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \cup, [\,], \neg))$ *is in PTIME under fixed, nonrecursive DTDs. Furthermore,* $\mathsf{SAT}(\mathcal{X}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \cup, [\,], =, \neg))$ *is in PTIME if the nonrecursive DTDs do not contain Kleene star.*

PROOF. First, note that if the fixed nonrecursive DTD D does not have Kleene star in it, there are at most $|D|^{(|D|^{|D|})}$ many tree instances of D, each bounded by $|D|^{|D|}$ in size. But $|D|^{(|D|^{|D|})}$ and $|D|^{|D|}$ are constants when D is fixed. Thus, a simple algorithm to check the satisfiability of a query p in $\mathcal{X}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \cup, [\,], =, \neg)$ is to evaluate p over each instance of D. Over each instance, a query p can be evaluated in PTIME in $|p|$ [Gottlob et al. 2005], since $|D|^{|D|}$ is a constant. Thus, the algorithm decides in PTIME whether or not p is satisfiable by any instance of D, and hence

whether or not (p, D) is satisfiable. In particular, for $\mathcal{X}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \cup, [\ ], \neg)$, it is in linear time [Gottlob et al. 2005].

We now show that $\mathsf{SAT}(X(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \cup, [\ ], \neg))$ is in PTIME over fixed nonrecursive DTDs. To do this, we show that given a (without loss of generality normalized) fixed nonrecursive DTD D, there is a nonrecursive DTD D' such that D' does not use Kleene star, it has the same root tag as D and moreover, for any query $p \in X(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \cup, [\ ], \neg)$, (p, D) is satisfiable if and only if (p, D') is satisfiable. The result now follows from the special case above.

We say that two trees T_1 and T_2 are *D-equivalent* if they agree on all queries in $X(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \cup, [\ ], \neg)$ that mention only element types and attributes of D. We claim:

CLAIM 6.5. *There is an integer function $g(n)$ such that for any tree T_1 conforming to a nonrecursive DTD D of size n, there is a tree T_2 such that T_2 conforms to D, it is D-equivalent to T_1, and moreover, every node in T_2 has at most $g(n)$ children.*

This claim suffices. For if it holds, we can proceed to transform D to to a DTD D' which does not contain Kleene star, by replacing productions of the form $A \to B^*$ with $A \to (\varepsilon + B + \cdots + B^{g(n)})$, where B^i abbreviates the i-fold concatenation of B. Since n is a constant, so is $g(n)$. Then, for any p in $X(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \cup, [\ ], \neg)$, (p, D) is satisfiable if and only if there exists a tree T_2 as described in the claim that satisfies p if and only if (p, D') is satisfiable.

We prove Claim 6.5 by induction on n. Let k be the depth of D (i.e, the size of the longest path in the DTD graph of D starting from the root element type). If D has only a root, then this is clear. The induction case is also straightforward when the root production is not of the form $r \to B^*$. So we consider the case when D has a root production of the above form. Let $D(r)$ be the DTD obtained by removing r and setting B as the root. Note that $|D(r)| \leq n - 1$, that is, the size of $D(r) \leq n - 1$. Let m be the number of isomorphism types of trees with depth $k - 1$, with only attributes mentioned in D and branching at most $g(n - 1)$. We set $g(n) = \mathsf{max}\{m, g(n-1)\}$. Now given T_1 conforming to D, we classify the subtrees of the root according to $D(r)$-equivalence. We transform T_1 to T_2 by keeping at most one subtree in each $D(r)$-equivalence class, and for any such representative subtree we replace it by a $D(r)$-equivalent one with branching at most $g(n - 1)$. Clearly, T_2 has the required branching. One can now easily show that T_2 is indeed D-equivalent to T_1. □

Unfortunately, fixed DTDs do not make our lives much easier: all the fragments studied in Propositions 4.2 and 4.3 remain intractable under fixed DTDs. These results require more involved encoding arguments than the ones given in Section 4, since the former relied heavily on varying DTDs.

THEOREM 6.6. *Under fixed DTDs, the following satisfiability problems are NP-hard:* (1) $\mathsf{SAT}(\mathcal{X}(\cup, [\ ]))$, (2) $\mathsf{SAT}(\mathcal{X}(\downarrow, [\ ]))$, *and* (3) $\mathsf{SAT}(\mathcal{X}(\downarrow, \uparrow))$.

PROOF. We prove these by reduction from 3SAT. Consider an instance $\phi = C_1 \wedge \cdots \wedge C_n$ of 3SAT, and assume that $x_1, \ldots, x_m$ are all the propositional variables used in ϕ.

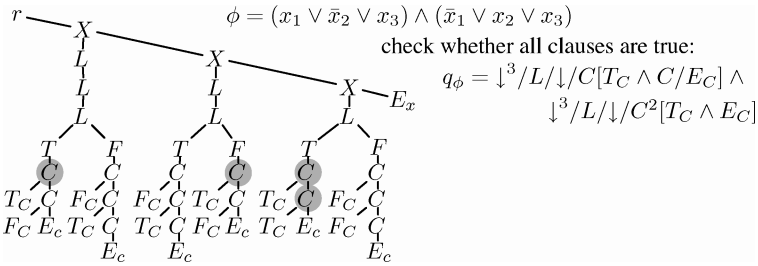

FIG. 6. A tree of the DTD encoding a 3SAT instance in the proof of case (2) in Theorem 6.7. The gray discs indicate which clauses are satisfied; in the rightmost branch both clauses are satisfied.

1. $\mathsf{SAT}(\mathcal{X}(\cup, [\,]))$ *is NP-hard.* We first define the fixed DTD $D_0 = (Ele, Att, P, R, r)$:

$Ele = \{X, T, F\} \cup \{r\}$.
P: $r \to X$, $\quad X \to (X + \varepsilon), (T + F)$.
$Att = \emptyset$, $\quad R(A) = \emptyset$ for all $A \in Ele$.

An XML tree of D_0 consists of a chain of X elements. The i-fold concatenation $X/\cdots/X$ of X starting from the root, denoted by X^i, encodes variable x_i. Each X element has either a T or F child, which encodes a unique truth value of the corresponding variable x_i. Note that D_0 is independent of ϕ.

We then encode the 3SAT instance ϕ in terms of a query $\mathsf{XP}(\phi) \in \mathcal{X}(\cup, [\,])$, defined with the following qualifiers at the root.

—Encoding variables: For each variable x_i in ϕ, let $\mathsf{XP}(x_i) = X^i/T$ and $\mathsf{XP}(\bar{x}_i) = X^i/F$.

—Encoding clauses: For each clause C_j we let $\mathsf{XP}(C_j)$ be C_j in which each x_i is replaced by $\mathsf{XP}(x_i)$ and each $\bar{x}_i$ is replaced by $\mathsf{XP}(\bar{x}_i)$. For example, if $C = x_i \vee \bar{x}_j \vee x_k$, then $\mathsf{XP}(C) = X^i/T \cup X^j/F \cup X^k/T$.

Finally, we use $\mathsf{XP}(\phi) = [\mathsf{XP}(C_1) \wedge \cdots \wedge \mathsf{XP}(C_n)]$ to encode ϕ, which simply checks whether all qualifiers $\mathsf{XP}(C_j)$, $j \in [1, n]$, are satisfiable.

It is easy to verify that a tree T of D_0 satisfies $\mathsf{XP}(C)$ iff the clause C is satisfiable, and furthermore, that ϕ is satisfiable if and only if there exists an XML tree $T \models (\mathsf{XP}(\phi), D_0)$.

2. $\mathsf{SAT}(\mathcal{X}(\downarrow, [\,]))$ *is NP-hard.* Consider a fixed DTD $D_1 = (Ele, Att, P, R, r)$ defined as:

$Ele = \{r,\ X,\ L,\ C,\ T,\ F,\ T_C,\ F_C,\ E_x, E_c\}$.
P: $r \to X + E_x$, $\quad X \to L, (X + E_x)$, $\quad L \to L + (T, F)$, $\quad C \to (T_C + F_C), (C + E_c)$,
$T \to C$, $\quad F \to C$, $\quad E_x \to \varepsilon$, $\quad E_c \to \varepsilon$,
$T_C \to \varepsilon$, $\quad F_C \to \varepsilon$.
$Att = \emptyset$, $R = \emptyset$.

As shown in Figure 6, an XML tree of D_1 consists of a chain of X elements. Below each X element is a chain of L elements, which may end with T and F children; and below each T and each F is a chain of C elements, while each C element may have either a T_C or a F_C child. The elements E_x, E_c indicate the end of the X and C chains, respectively. Note that D_1 is independent of ϕ.

We encode the 3SAT instance ϕ in terms of a query $\mathsf{XP}(\phi)$ in $\mathcal{X}(\downarrow, [\,])$, defined with the following qualifiers at the root.

—Encoding variables: $q_v = X^m[E_x]$. This asserts that the X chain consists of precisely m elements of type X. We use X^j to encode the variable x_j in ϕ.

—Coding the connection between clauses and literals:

$$q_c = \bigwedge_{i\in[1,n],\, j\in[1,m]} \left(q^T_{i,j} \wedge q^F_{i,j}\right),$$

where $q^T_{i,j}, q^F_{i,j}$ are defined as follows:

$$q^T_{i,j} = \begin{cases} X^j/L^{m-j+1}/T/C^i/T_C & \text{if } x_j \text{ appears in } C_i \\ X^j/L^{m-j+1}/T/C^i/F_C & \text{otherwise} \end{cases}$$

$$q^F_{i,j} = \begin{cases} X^j/L^{m-j+1}/F/C^i/T_C & \text{if } \bar{x}_j \text{ appears in } C_i \\ X^j/L^{m-j+1}/F/C^i/F_C & \text{otherwise} \end{cases}$$

That is, we encode the clause C_i in terms of C^i under T and F of each X, where C^i is a shorthand for $C/\ldots/C$ of length i. For each variable x_j (i.e., X^j), if x_j appears (positively) in C_i, then $q^T_{i,j}$ ensures that C^i under T has a T_C child, that is, C_i is satisfied if x_j is true; if x_j does not appear in C_i, then C^i under T has a F_C child; similarly, $q^F_{i,j}$ encodes the connection between $\bar{x}_j$ (i.e., x_j appears negatively) and C_i.

Observe that the qualifiers assert that below X_j, the L chain is of length $m - j + 1$. That is, each L chain must end up having T and F children after going downward $m - j + 1$ times following L. Intuitively, the distance from the root to the truth assignment of the X element coding x_j (i.e., X^j) is precisely $m + 2$. As a result, the distance between the root and each C_i clause (i.e., C^i) is precisely $m + i + 2$, no matter under which x_j (i.e., X_j) the C^i element appears.

—Coding a consistent truth assignment:

$$q_a = \bigwedge_{j\in[1,m]} (X^j[L^{m-j+1}/\downarrow/C^n/E_c \;\wedge\; L^{m-j+1}/\downarrow/C^{n+1}/E_c]).$$

We encode x_j (i.e., X^j) such that it is assigned true if the C chain under the T child of X^j has precisely n elements of C type; similarly, x_j is false if the C chain under the F child of X^j has n elements of C type. The qualifier q_a asserts that for each x_j there is a single truth value, that is, only one of the C chains under X^j has n elements of type C.

—Encoding clauses:

$$q_\phi = \bigwedge_{i\in[1,n]} \downarrow^m/L/\downarrow/C^i[T_C \wedge C^{n-i}/E_c].$$

This asserts that any clause C_i (e.g., C^j) must be satisfied by a truth assignment of some x_j, no matter what x_j is.

Taken together, the query $\mathsf{XP}(\phi)$ is defined to be $\varepsilon[q_v \wedge q_c \wedge q_a \wedge q_\phi]$. One can easily verify that $\mathsf{XP}(\phi)$ is in $\mathsf{SAT}(\mathcal{X}(\downarrow, [\,]))$ and furthermore, that $\mathsf{XP}(\phi)$ is satisfiable by an XML tree of the fixed DTD D_1 iff ϕ is satisfiable.

3. $\mathsf{SAT}(\mathcal{X}(\downarrow, \uparrow))$ *is NP-hard.* It has been shown in Benedikt et al. [2005] that for each query p in $\mathcal{X}(\downarrow, [\,])$, if p does not contain label test ($\mathsf{lab}() = A$), then it can

be rewritten into a query $\mathsf{rewrite}(p)$ in $\mathcal{X}(\downarrow, \uparrow)$ such that for any XML tree T, $T \models p$ iff $T \models \mathsf{rewrite}(p)$. Observe that the query $\mathsf{XP}(\phi)$ used in the NP-hardness proof of Case (2) above does not contain any label test. Thus, it can be rewritten into an equivalent query in $\mathcal{X}(\downarrow, \uparrow)$. As an immediate result, the same fixed DTD used in Case (2) and the rewritten query in $\mathcal{X}(\downarrow, \uparrow)$ yield a reduction from 3SAT to $\mathsf{SAT}(\mathcal{X}(\downarrow, \uparrow))$.

More specifically, there is a linear-time function $\mathsf{rewrite}(p)$ transforming queries (without label test) in $\mathcal{X}(\downarrow, [\,])$ into an equivalent query in $\mathcal{X}(\downarrow, \uparrow)$. The rewriting is conducted inductively on the structure of p, using the following rules.

(1) $\mathsf{rewrite}(\eta) = \eta$, when η is A, $\downarrow$ or ε.
(2) $\mathsf{rewrite}(p_1/p_2) = \mathsf{rewrite}(p_1)/\mathsf{rewrite}(p_2)$.
(3) $\mathsf{rewrite}(p_1[q]) = \mathsf{rewrite}(p_1)/\mathsf{rewrite}([q])$.
(4) $\mathsf{rewrite}([\varepsilon]) = \varepsilon$.
(5) $\mathsf{rewrite}([\eta]) = \eta/\uparrow$, when η is A or $\downarrow$.
(6) $\mathsf{rewrite}([p_1/p_2]) = \mathsf{rewrite}([p_1])/\mathsf{rewrite}([p_2])$.
(7) $\mathsf{rewrite}([q_1 \wedge q_2) = \mathsf{rewrite}([q_1])/\mathsf{rewrite}([q_2])$.

This rewriting takes $O(|p|)$ time. By means of $\mathsf{rewrite}$ one can transform the query $\mathsf{XP}(\phi)$ used in the proof of Case (2) into an equivalent query $\mathsf{rewrite}(\mathsf{XP}(\phi))$ in $\mathcal{X}(\downarrow, \uparrow)$. □

The next result shows that the PSPACE (Theorem 5.2) and the EXPTIME (Theorems 5.3, 5.6) lower bounds remain intact under fixed DTDs; and worse still, so does the undecidability (Theorem 5.4).

THEOREM 6.7. *Under fixed DTDs,*

(1) $\mathsf{SAT}(\mathcal{X}(\downarrow, [\,], \neg))$ *is PSPACE-hard;*
(2) $\mathsf{SAT}(\mathcal{X}(\downarrow, \downarrow^*, [\,], \neg))$ *is EXPTIME-hard;*
(3) $\mathsf{SAT}(\mathcal{X}(\uparrow, [\,], =, \neg))$ *is EXPTIME-hard;*
(4) $\mathsf{SAT}(\mathcal{X}(\downarrow, \uparrow, \downarrow^*, \uparrow^*, \cup, [\,], =, \neg))$ *is undecidable.*

PROOF. We prove these results by reduction from 3QSAT, the two player corridor tiling game and the halting problem for two-register machines, respectively.

1. $\mathsf{SAT}(\mathcal{X}(\downarrow, [\,], \neg))$ *is PSPACE-hard.* We prove this by reduction from Q3SAT. We define the fixed DTD $D_0 = (Ele, Att, P, R, r)$ as follows:

$Ele = \{X, T, F\}$.
P: $\quad r \to X, \quad X \to T^*, F^*, \quad T \to X, \quad F \to X$.
$Att = \emptyset, \quad R(A) = \emptyset$ for all $A \in Ele$.

Compared to the DTD defined in the proof of Proposition 5.1, in the DTD D_0 (a) we encode the variable x_i by means of $\downarrow^{2(i-1)}/X$; and (b) we encode the universal and existential quantifications uniformly in terms of T^*, F^*, instead of T, F (universal) and $T + F$ (existential) as in the proof of Proposition 5.1; as will be seen shortly, we use XPath qualifiers to distinguish between the two cases.

Consider an instance of the Q3SAT problem, that is, a quantified Boolean sentence $\phi = Q_1 x_1 Q_2 x_2 \cdots Q_m x_m \phi$, where $\phi = C_1 \wedge \cdots \wedge C_n$ is an instance of 3SAT and each $Q_i \in \{\forall, \exists\}$. We use the following qualifiers to encode ϕ.

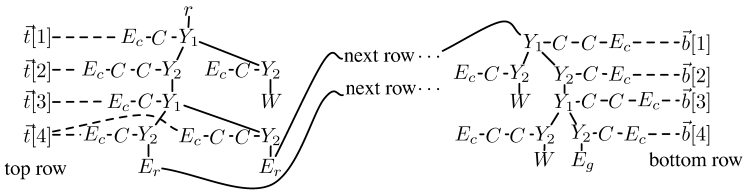

FIG. 7. An XML tree of the DTD encoding of TPG-CT in the proof of Theorem 6.7(2). Dashed lines represent that the vertical conditions are satisfied.

—Encoding the quantifiers: For each quantifier $Q_i x_i$, we define a qualifier q_i ensuring that all X elements reached by $\downarrow^{2(i-1)}/X$ have both T and F children if $Q_i = \forall$. That is,

$$q_i = \neg\downarrow^{2(i-1)}/X[\neg(T \wedge F)].$$

Similarly, in case $Q_i = \exists$, we use q_i to ensure that $\downarrow^{2(i-1)}/X$ has either T or F child:

$$q_i = \neg\downarrow^{2(i-1)}/X[T \wedge F].$$

—Encoding the clauses: For each clause $C_j = l_j^1 \vee l_j^2 \vee l_j^3$, where l_j^i is a literal, that is, it is either a variable x_s or the negation $\bar{x}_s$ of a variable, we define $\mathsf{XP}(C_j)$ to encode the negation of C_j, that is, $\neg l_j^1 \wedge \neg l_j^2 \wedge \neg l_j^3$. Without loss of generality, we may assume that the variables of the literals are x_s, x_t and x_u with $s < t < u$. Then

$$\mathsf{XP}(C_j) = \downarrow^{2s-2}/X/Z_s/\downarrow^{2(t-s)-2}/X/Z_t/\downarrow^{2(u-t)-2}/X/Z_u,$$

where $Z_i = F$ if x_i appears in C_j and $Z_i = T$ if $\bar{x}_i$ appears in C_j, when $i \in \{s, t, u\}$.

Finally, we express that none of the negated clauses is satisfied

$$\mathsf{XP}(\phi) = \varepsilon\left[\bigwedge_{j\in[1,n]} \neg\mathsf{XP}(C_j) \wedge \bigwedge_{i\in[1,m]} q_i\right].$$

It is easy to verify that ϕ is satisfiable if and only if $\mathsf{XP}(\phi)$ is satisfiable over the fixed D_0.

2. $\mathsf{SAT}(\mathcal{X}(\downarrow, \downarrow^*, [\,], \neg))$ *is EXPTIME-hard.* We show this by a reduction to TPG-CT (Two-Player Game of Corridor Tiling), where TPG-CT is stated in the proof of Theorem 5.6. We first define a fixed DTD $D_1 = (Ele, Att, P, R, r)$ as follows:

$Ele = \{Y_1, Y_2, C, W, L, E_c, E_r, E_g\}$.

P: $r \to Y_1$, $\quad Y_1 \to C, (Y_2^* + L)$, $\quad Y_2 \to (C, (Y_1 + E_r + E_g + W))$

$W \to W + E_r + E_g$, $\quad L \to L + E_r + E_g$, $\quad E_r \to Y_1 + W + L$,

$E_g \to \varepsilon$, $\quad C \to C + E_c$, $\quad E_c \to \varepsilon$.

$Att = \emptyset$, $\quad R(A) = \emptyset$ for all $A \in Ele$.

An XML tree of the DTD D_1 is depicted in Figure 7. We use Y_1 and Y_2 to represent the moves made by Player I and Player II, respectively. Under each Y_1 (respectively, Y_2) element there exists a chain of C elements whose length is to encode the tile placed in the move by Player I (respectively, Player II). We use W, L to indicate that Player I wins or loses the game, and E_r, E_g to indicate the end of a row and the game, respectively.

Given an instance $(X, H, V, \vec{t}, \vec{b})$ and n of TPG-CT, we encode the instance in terms of the DTD D_1 and an XPath query Q_1 in $\mathcal{X}(\downarrow, \downarrow^*, [\,], \neg)$ such that there exists an XML tree T satisfying (Q_1, D_1) if and only if Player I has a winning strategy for the game. To simplify the presentation, we first give Q_1 in terms of qualifiers in $\mathcal{X}(\downarrow, \downarrow^*, \cup, [\,], \neg)$, and then show that union (disjunction) can be eliminated by using negation and conjunction. We assume, without loss of generality, that n is even and Player I moves first; the coding for an odd n is similar.

(1) *Game tree*. To encode a winning strategy for Player I, we want the game tree to represent the alternating plays of Player I (who chooses a single tile) and Player II (who tries all tiles). We use a C chain $C^i = C/\cdots/C$ (i times) under Y_1 (respectively, Y_2) to represent a tile x_i in $X = \{x_1, \ldots, x_k\}$ placed by Player I (respectively, Player II). To ensure that the tile is in X, we use a qualifier Q_{one}:

$$Q_{one} = \neg(\downarrow^*/Y_1[C^{k+1}]) \ \wedge \ \neg(\downarrow^*/Y_2[C^{k+1}]).$$

For the moves by Player II, we use Q_{one} to assure that all possible k tiles are tried:

$$Q_{all} = \neg\left(\downarrow^*/Y_1\left[\neg \bigwedge_{i\in[1,k]} Y_2/C^i/E_c\right]\right).$$

(2) Every row consists of exactly n tiles: no row consists of fewer or more than n tiles.

$$Q^n = \neg\left(\downarrow^*/(r \ \cup \ E_r)\left[\bigvee_{i\in[1,n-1]} \Sigma^i[E_r \cup E_g] \ \vee \ \Sigma^{n+1}\right]\right),$$

where $\Sigma = Y_1 \cup Y_2 \cup W \ \cup \ L$ and Σ^i denotes $\Sigma/\cdots/\Sigma$ (i times).

Furthermore, L can only be followed by L, E_r, E_g; similarly for W:

$$Q_{(w,l)} = \neg(\downarrow^*/L[\downarrow^*[\neg(L \vee E_r \vee E_g)]]) \ \wedge \ \neg(\downarrow^*/W[\downarrow^*[\neg(W \vee E_r \vee E_g)]]).$$

(3) Player I places an invalid tile if and only if the move is followed by a chain of L (and E_r, E_g). That is, a move Y_1 has an L child if and only it violates the adjacency conditions. More specifically, if a tile is not in the top row, then it violates one of the vertical, horizontal or bottom conditions; otherwise, it violates either the horizontal condition or the top row. This is encoded with Q_I, which is the conjunction of the qualifiers given below.

(i) If a move Y_1 has an L child then it violates the adjacency conditions.

$$\begin{aligned} Q_L \ = \neg\big(\downarrow^*\big[\big((\Sigma \cup E_r)^{n+1}[L] \ \wedge \ &\left(\bigvee_{(i,j)\in V} Y_1[C^i/E_c]/(\Sigma \cup E_r)^{n+1}[Y_1[C^j/E_c]]\right) \ \wedge \\ &\left(\bigvee_{(i,j)\in H} (\Sigma \cup E_r)^n[Y_2[C^i/E_c]/Y_1[C^j/E_c]]\right) \ \wedge \\ &\bigvee_{l\in\{1,3,\ldots,n-1\}} \left((\Sigma \cup E_r)^{n+1}[[(\Sigma \cup E_r)^{n-l+1}[E_g]] \bigvee_{(i,\vec{b}[l])\in V} Y_1[C^i/E_c]])\right]\right) \ \wedge \\ \neg\Bigg(\bigvee_{l\in\{1,3,\ldots,n-1\}} \Sigma^l[L] \ \wedge \ &(\Sigma^{l-2}[\bigvee_{(i,j)\in H} Y_2[C^i/E_c]/Y_1[C^j/E_c]]\Big) \ \wedge \\ &\left(\Sigma^{l-1}\left[\bigvee_{(\vec{t}[l],i)\in V} Y_1[C^i/E_c]]\right)\right)\Bigg). \end{aligned}$$

(ii) If Player I makes a move violating the horizontal constraints of H, then the node representing the move has a child L:

$$Q_{(1,h)} = \neg\left(\downarrow^*\left[\bigvee_{(i,j)\in(X\times X)\setminus H}(Y_2[C^i/E_c]/Y_1[C^j/E_c]\vee Y_2[C^i/E_c]/E_r/Y_1[C^j/E_c])[\neg L]\right]\right).$$

(iii) Furthermore, if Player I makes a move violating the vertical constraints of V, then the node representing the move has a child L:

$$Q_{(1,v)} = \neg\left(\downarrow^*\left[\bigvee_{(i,j)\in(X\times X)\setminus V} Y_1[C^i/E_c]/(\Sigma\cup E_r)^{n+1}[Y_1[C^j/E_c][\neg L]]\right]\right).$$

(iv) Moreover, if Player I makes a move violating the top row $\vec{t}$, then the node representing the move has a child L:

$$Q_{(1,t)} = \neg\left(\bigvee_{l\in\{1,3,\ldots,n-1\}}\bigvee_{(\vec{t}[l],i)\in(X\times X)\setminus V}\Sigma^{l-1}[Y_1[C^i/E_c][\neg L]]\right).$$

(v) Similarly, if Player I makes a move violating the bottom row $\vec{b}$, then the node representing the move has a child L:

$$Q_{(1,b)} = \neg\left(\downarrow^*\left[\bigvee_{l\in\{1,3,\ldots,n-1\}}\bigvee_{(i,\vec{b}[l])\in(X\times X)\setminus V} Y_1[C^i/E_c][\neg L]\,[\Sigma^{n-l}[E_g]]\right]\right).$$

Here, $Q_I = Q_L \wedge Q_{(1,h)} \wedge Q_{(1,v)} \wedge Q_{(1,t)} \wedge Q_{(1,b)}$.

Similarly, one can code Q_{II} for Player II in terms of W.

Putting these together, the coding of the TPG-CT system consists of the DTD D_1 and the query Q_1 in $\mathcal{X}(\downarrow, \downarrow^*, \cup, [\,], \neg)$:

$$Q_1 = \varepsilon[\neg\downarrow^*/L \wedge Q_{one} \wedge Q_{all} \wedge Q^n \wedge Q_{w,l} \wedge Q_I \wedge Q_{II}].$$

It is easy to verify that Player I has a winning strategy if and only there is a finite XML tree T, which represents a game tree, such that $T \models (Q_1, D_1)$.

We remark that, in the presence of label tests and negation, we can eliminate union and disjunction in Q_1. Indeed, disjunction can be easily represented in terms of conjunction and negation, and Σ in the coding can be replaced by:

$$\widetilde{\Sigma} = \downarrow[\neg(\neg\mathsf{lab}() = Y_1 \wedge \neg\mathsf{lab}() = Y_2 \wedge \neg\mathsf{lab}() = W \wedge \neg\mathsf{lab}() = L)].$$

Similarly, we rewrite Σ^i into the union-free expression $\widetilde{\Sigma}^i$.

3. $\mathsf{SAT}(\mathcal{X}(\uparrow, [\,], =, \neg))$ *is EXPTIME-hard.* The proof is a mild variation of that of Theorem 5.6, by reduction from TPG-CT. We first define a fixed DTD that does not depend on any instance of TPG-CT. Let $D_2 = (Ele, Att, P, R, r)$, where

$Ele = \{r, C, X\}$.
P: $\quad r \to C^*, \qquad C \to X, \qquad X \to X + \varepsilon.$
$Att = \{@t, @h, @k, @\mathsf{next}\}$,
R: $\quad R(C) = \{@h, @k, @\mathsf{next}\}, \quad R(X) = \{@t\}, \qquad R(r) = \emptyset.$

An XML tree of D_2 consists of a root node and its C children. Each C node has a chain of X elements below it. In a nutshell, as in the proof of Theorem 5.6, each

C element v encodes a snapshot of the game showing the last n plays, where n is the length of a row in the corridor; the attribute $v.h$ encodes the horizontal position of the last tile $v.t_n$ in a row, and $v.k$ and $v.\mathsf{next}$ encode a list of such snapshots. However, in contrast to $v.t_i$ in the proof of Theorem 5.6, here, we use $X^i/@t$ below v to encode the ith tile in the snapshot, where X^i abbreviates the ith fold concatenation of X.

Given an instance $(X, H, V, \vec{t}, \vec{b})$ and n of TPG-CT, we encode the instance in terms of the DTD D_2 and a query Q_2 in $\mathcal{X}(\uparrow, [\,], =, \neg)$ such that there exists an XML tree T satisfying (Q_2, D_2) if and only if Player I has a winning strategy for the game. The query Q_2 is a mild variation of Q_0 given in the proof of Theorem 5.6, by (1) substituting $X^i/@t$ for each occurrence of attribute $@t_i$ in Q_0, for $i \in [1, n]$, and (2) adding an additional conjunct to the set of qualifiers of Q_0 at the root: $Q_t = \neg C[\neg X^n]$. Intuitively, Q_t asserts that every C element v has a chain of X elements below it, and the chain has a length no less than n. Thus, one can use $X^i/@t$ below v to encode the ith tile in the snapshot represented by v. The rest of the proof is the same as its counterpart for Theorem 5.6.

4. $\mathsf{SAT}(\mathcal{X}(\downarrow, \uparrow, \downarrow^*, \uparrow^*, \cup, [\,], =, \neg))$ *is undecidable.* Observe that the DTD used in the proof of Theorem 5.4 is already fixed: it is independent of 2RM instances. Thus we have already shown that $\mathsf{SAT}(\mathcal{X}(\downarrow, \uparrow, \downarrow^*, \uparrow^*, \cup, [\,], =, \neg))$ is undecidable for fixed DTDs. □

6.3. DISJUNCTION-FREE DTDS. Recall that a DTD is disjunction-free if no productions in it contain disjunction '+' (Section 2). The fact that disjunction-free DTDs are easier to analyze was already noted in other contexts [Lakshmanan et al. 2004; Martens and Neven 2004]. The absence of disjunction makes the satisfiability analysis simpler for certain fragments. Below, we show that in contrast to Proposition 4.2, for a fragment that contains $\mathcal{X}(\downarrow, [\,])$ and $\mathcal{X}(\cup, [\,])$, the satisfiability analysis becomes tractable.

THEOREM 6.8. *Under disjunction-free DTDs, the following problems are in PTIME:* (1) $\mathsf{SAT}(\mathcal{X}(\downarrow, \downarrow^*, \cup, [\,]))$*;* (2) $\mathsf{SAT}(\mathcal{X}(\downarrow, \uparrow))$.

PROOF. We show that these problems are in PTIME under disjunction-free DTDs.

1. $\mathsf{SAT}(\mathcal{X}(\downarrow, \downarrow^*, \cup, [\,]))$ *is in PTIME.* We show this by providing an $O(|p||D|^2)$-time decision algorithm based on dynamic programming, along the same lines as the proof of Theorem 4.1. Let p be a query in $\mathcal{X}(\downarrow, \downarrow^*, \cup, [\,])$ and D be a disjunction-free DTD. We refer to the proof of Theorem 4.1 for the definition of the DTD graph G_D and the construction of the list $\mathcal{L}$ of all subqueries of p in "ascending" order.

To present the algorithm, we first define variables used in the algorithm. For each $p' \in \mathcal{L}$ and each element type $A \in D$, we define two variables:

—$\mathsf{reach}(p', A)$: the set of element types reachable from A via p' in G_D.

—$\mathsf{sat}(p', A)$: the truth value indicating whether or not p' is satisfiable at A.

Note that since we do not have data values, the satisfiability of p' at an A element can be determined from $\mathsf{reach}(p', A)$ alone.

The key observation is that for disjunction-free DTDs, the conjunction of qualifiers $[q_1 \wedge \cdots \wedge q_n]$ is satisfiable at an A element, that is, $\mathsf{sat}([q_1 \wedge \cdots \wedge q_n], A)$, if and

only if for all i, $\mathsf{sat}([q_i], A)$, that is, the truth value of $q_1 \wedge \cdots \wedge q_n$ can be determined by checking the truth values of each q_i independently. This follows directly from the fact that we only have two kinds of nontrivial productions: $A \to B_1, \ldots, B_k$ and $A \to B^*$. Note that this fact about $[q_1 \wedge \cdots \wedge q_n]$ is no longer true either in the presence of productions of the form $A \to B_1 + \cdots + B_k$, or in the presence of negation (or data values) in the XPath fragment.

The decision algorithm is outlined as follows:

(1) For each $p' \in \mathcal{L}$ (in the order of $\mathcal{L}$) and $A \in Ele$, we compute $\mathsf{reach}(p', A)$ and $\mathsf{sat}(p', A)$, based on the structure of p':
   (a) $p' = \varepsilon$: then $\mathsf{reach}(p', A) = \{A\}$;
   (b) $p' = l$: then $\mathsf{reach}(p', A) = \{l\}$ if $P(A)$ contains l;
   (c) $p' = \downarrow$: then $\mathsf{reach}(p', A)$ is the set of element types in $P(A)$;
   (d) $p' = \downarrow^*$: then $\mathsf{reach}(p', A)$ is the set of element reachable from A in G_D;
   (e) $p' = p_1 \cup p_2$: then $\mathsf{reach}(p', A) = \mathsf{reach}(p_1, A) \cup \mathsf{reach}(p_2, A)$;
   (f) $p' = p_1/p_2$: then $\mathsf{reach}(p', A) = \bigcup_{B \in \mathsf{reach}(p_1, A)} \mathsf{reach}(p_2, B)$.
   In all the cases above, $\mathsf{sat}(p', A) = \mathsf{true}$ if and only if $\mathsf{reach}(p', A) \neq \emptyset$.
   (g) $p' = \varepsilon[q]$: $\mathsf{sat}(p', A) = \mathsf{sat}(q, A)$, and $\mathsf{reach}(p', A) = \{A\}$ if $\mathsf{sat}(q, A) = \mathsf{true}$;
   (h) $p' = [p_1]$: then $\mathsf{sat}(p', A) = \mathsf{sat}(q, A)$;
   (i) $p' =$ '$\mathsf{lab}() = A$': then $\mathsf{sat}(p', A) = \mathsf{true}$ and $\mathsf{sat}(p', B) = \mathsf{false}$ for all other B;
   (j) $p' = [q_1 \wedge \cdots \wedge q_n]$: then $\mathsf{sat}(p', A) = \mathsf{sat}([q_1], A) \wedge \cdots \wedge \mathsf{sat}([q_n], A)$;
   (k) $p' = [q_1 \vee \cdots \vee q_n]$: then $\mathsf{sat}(p', A) = \mathsf{sat}([q_1], A) \vee \cdots \vee \mathsf{sat}([q_n], A)$.

(2) Return $\mathsf{sat}(p, r)$, where r is the root type of D.

Note that since $p[q] = p/\varepsilon[q]$, we can reduce the inductive case for $p[q]$ to p_1/p_2 and $\varepsilon[q]$.

This gives us an $O(|p||D|^2)$-time algorithm. The correctness of the algorithm can be verified in a way similar to the proof of Theorem 4.1. The only difference is the treatment of qualifiers, for which the soundness follows from the observation described above.

2. $\mathsf{SAT}(\mathcal{X}(\downarrow, \uparrow))$ *is in PTIME.* We show that $\mathsf{SAT}(\mathcal{X}(\downarrow, \uparrow))$ is in $O(|p||D|^2)$-time, by reducing $\mathsf{SAT}(\mathcal{X}(\downarrow, \uparrow))$ to $\mathsf{SAT}(\mathcal{X}(\downarrow, [\ ]))$. Let D be a disjunction-free DTD and p a query be in $\mathsf{SAT}(\mathcal{X}(\downarrow, \uparrow))$. We rewrite p into an equivalent (at the root) query $\mathsf{rewrite}(p)$ in $\mathcal{X}(\downarrow, [\ ])$ such that for any XML tree T, $T \models (p, D)$ iff $T \models (\mathsf{rewrite}(p), D)$. We compute $\mathsf{rewrite}(p)$ in linear-time, inductively on the structure of p, by extending the rewriting rules introduced in Benedikt et al. [2005], as follows.

(1) $\mathsf{rewrite}(\eta) = \eta$, where η is one of ε, A, $\downarrow$ or $\uparrow$.
(2) $\mathsf{rewrite}(p'/\eta/\uparrow) = \mathsf{rewrite}(p')[\eta]$, where η is one of ε, A or $\downarrow$.
(3) $\mathsf{rewrite}(p'/\uparrow/\uparrow) = \mathsf{rewrite}(\mathsf{rewrite}(p'/\uparrow)/\uparrow)$.
(4) $\mathsf{rewrite}(p'/\eta'/\eta) = \mathsf{rewrite}(p'/\eta')/\eta$, where η is one of ε, A or $\downarrow$, and η' is one of ε, A, $\downarrow$ or $\uparrow$.

If $\mathsf{rewrite}(p)$ contains $\uparrow$, then it is not satisfiable. Otherwise, the $O(|p||D|^2)$-time decision algorithm $\mathsf{SAT}(\mathcal{X}(\downarrow, \downarrow^*, \cup, [\,]))$ given above can be applied to $\mathsf{rewrite}(p)$. □

However, once we extend $\mathcal{X}(\cup, [\,])$ and $\mathcal{X}(\downarrow, [\,])$ by adding data values $(=)$ or upward axes, the disjunction-free distinction no longer affects the worst-case behavior.

THEOREM 6.9. *Under disjunction-free DTDs, the following problems are NP-hard: (1)* $\mathsf{SAT}(\mathcal{X}(\cup, [\,], =))$*; (2)* $\mathsf{SAT}(\mathcal{X}(\downarrow, [\,], =))$*; (3)* $\mathsf{SAT}(\mathcal{X}(\downarrow, \uparrow, \cup, [\,]))$*. The last result holds under fixed, disjunction-free DTDs.*

PROOF. We show these by reduction from 3SAT. Consider an instance $\phi = C_1 \wedge \cdots \wedge C_n$ of 3SAT, where $C_j = l_1^j \vee l_2^j \vee l_3^j$ and $l_s^j = x_i$ or $l_s^j = \bar{x}_i$ for some $i \in [1, m]$.

1. $\mathsf{SAT}(\mathcal{X}(\cup, [\,], =))$ *is NP-hard.* Given ϕ, we first define a disjunction-free DTD $D_0 = (Ele, Att, P, R, r)$ as follows:

$Ele = \{X\} \cup \{r\}$.
P: $r \to X$. $\quad Att = \{@x_1, \ldots, @x_m\}$, $\quad R(X) = \{@x_1, \ldots, @x_m\}$.

We then define a query $\mathsf{XP}(\phi)$ in $\mathcal{X}(\cup, [\,], =)$, as follows.

—Literals: We encode the variables x_i using attributes $@x_i$ and define $\mathsf{XP}(l_s^j) = (\varepsilon/@x_i = 1)$ if $l_s^j = x_i$ and $\mathsf{XP}(l_s^j) = (\varepsilon/@x_i = 0)$ if $l_s^j = \bar{x}_i$.
—Clauses: We encode a clause $C_j = l_1^j \vee l_2^j \vee l_3^j$ as $\mathsf{XP}(C_j) = \mathsf{XP}(l_1^j) \vee \mathsf{XP}(l_2^j) \vee \mathsf{XP}(l_3^j)$.
—Truth assignments: To ensure that the attribute values encode a consistent truth assignment, we use the qualifier $Q_t = \bigwedge_{i \in [1,m]} (\varepsilon/@x_i = 1 \vee \varepsilon/@x_i = 0)$.

Putting these together, we define $\mathsf{XP}(\phi) = X\big[Q_t \wedge \bigwedge_{j \in [1,n]} \mathsf{XP}(C_j)\big]$ in $\mathcal{X}(\cup, [\,], =)$, which checks the consistency of truth assignments and whether all clauses are satisfied. It is easy to verify that ϕ is satisfiable iff $(\mathsf{XP}(\phi), D_0)$ is satisfiable.

2. $\mathsf{SAT}\mathcal{X}(\downarrow, [\,], =))$ *is NP-hard.* Given an instance ϕ of 3SAT, we define a disjunction-free DTD $D_1 = (Ele, Att, P, R, r)$ as follows:

$Ele = \{X, \bar{X}, L_1', L_2', L_3'\} \cup \{L_i \mid \in [1, m]\} \cup \{C_i \mid i \in [1, n]\} \cup \{r\}$.
P: $r \to C_1, \ldots, C_n, L_1, \ldots, L_m$, $\quad C_j \to L_1', L_2', L_3'$, $\quad L_i \to X, \bar{X}$.
$Att = \{@v\}$, $\quad R(X) = R(\bar{X}) = R(L_i') = \{@v\}$ for $i \in [1, m]$.

As shown in Figure 8, an XML tree of D_1 is a constant-depth tree with a C_j node for each clauses C_j and an L_i node for each variable x_i appearing in ϕ. Each C_j node has three children L_1', L_2' and L_3', encoding literals in C_j; the truth values of L_1', L_2' and L_3' are determined by attributes $X.v$ and its negation $\bar{X}.v$ under the corresponding L_1, L_2 and L_3.

We define the following $\mathcal{X}(\downarrow, [\,], =)$ qualifiers, at the root.

—*Truth Assignment*: To assert that the attribute values form a meaningful truth assignment, for each $i \in [1, m]$, we define $t_i = [L_i[\downarrow/@v = 1 \wedge \downarrow/@v = 0]]$. That is, either x_i is assigned 1 and $\bar{x}_i$ is 0, or the other way around.

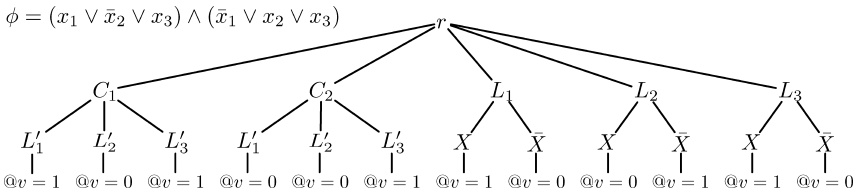

FIG. 8. An example of a tree T conforming to the DTD D_1, which is used in the proof of case (2) of Theorem 6.9.

—*Consistency*: The truth values should be assigned consistently across different clauses. For each $j \in [1, n]$ and $s \in [1, 3]$, where l_s^j is a literal of C_j, we define $q_j = [C_j/L'_s/@v = L_i/X/@v]$ if $l_s^j = x_i$, and $q_j = [C_j/L'_s/@v = L_i/\bar{X}/@v]$ if $l_s^j = \bar{x}_i$.

—*Encoding Clauses*: To encode clauses in ϕ, for each $j \in [1, n]$, we define $Q^j = [C_j/\downarrow/@v = 1]$, that is, one of its literals must be true.

Taken together, we encode the 3SAT instance by asserting that each clause must be true and that truth assignment is consistent: $\mathsf{XP}(\phi) = \varepsilon[\bigwedge_{j\in[1,n]}(Q^j \wedge q_j) \wedge \bigwedge_{i\in[1,m]} t_i]$. Then, it is easy to verify that $(\mathsf{XP}(\phi), D_1)$ is satisfiable iff ϕ is satisfiable.

3. $\mathsf{SAT}(\mathcal{X}(\downarrow, \uparrow, \cup, [\,]))$ *is NP-hard.* We show that $\mathsf{SAT}(\mathcal{X}(\downarrow, \uparrow, \cup, [\,]))$ is NP-hard under fixed, disjunction-free DTDs. We first define a fixed DTD $D_2 = (Ele, Att, P, R, r)$, where

$$Ele = \{T, F\} \cup \{r\}.$$
$$P: \quad r \to T^*, F^*, \qquad T \to T^*, F^*, \qquad F \to T^*, F^*.$$
$$Att = \emptyset, \qquad R(T) = R(F) = \emptyset.$$

An XML tree of D_2 has only T and F nodes. The chains of T and F are used to encode the variables. Although we have multiple chains due to presence of Kleene star, the upward modality ($\uparrow$) allow us to check the truth values of clauses along each chain.

Given an instance ϕ of 3SAT, we use $\mathcal{X}(\downarrow, \uparrow, \cup, [\,])$ qualifiers to encode ϕ as follows. Suppose that $x_1, \ldots, x_m$ are the variables in ϕ.

—Variables: We shall use an XPath query to ensure the existence of a chain consisting of T and F elements, such that the element at the $(i + 1)$-th position in the chain (reachable by $\downarrow^i$) encodes the truth value of x_i.

—Clauses: For each clause $C_j = l_1 \vee l_2 \vee l_3$, we define $q_j = \mathsf{XP}(l_1) \vee \mathsf{XP}(l_2) \vee \mathsf{XP}(l_3)$, where $\mathsf{XP}(l_i) = \uparrow^{(m-j)}[\mathsf{lab}() = T]$ if $l_i = x_j$ and $\mathsf{XP}(l_i) = \uparrow^{(m-j)}[\mathsf{lab}() = F]$ if $l_i = \bar{x}_j$,

Finally, we define $XP(\phi) = \varepsilon[\downarrow^{m+1}[q_1 \wedge \cdots \wedge q_n]]$ in $\mathcal{X}(\downarrow, \uparrow, \cup, [\,])$, which assures the existence of a chain of truth values satisfying ϕ. Observe that $\mathsf{XP}(\phi)$ first goes down $m + 1$ steps via $\downarrow$ to scan all variables and then goes upward from there via $\uparrow$ to check the satisfiability of ϕ. One can verify that $(\mathsf{XP}(\phi), D_2)$ is satisfiable iff ϕ is satisfiable. Note that D_2 is fixed and disjunction-free. □

The absence of disjunction in DTDs also has little impact on fragments with negation: the PSPACE and EXPTIME lower bounds are robust under

disjunction-free DTDs. This is because one can encode much of the semantics of disjunction in terms of a combination of the Kleene star in a DTD and XPath qualifiers with negation, as shown in the proof of the next result.

COROLLARY 6.10. *Under disjunction-free DTDs,*

(1) $\mathsf{SAT}(\mathcal{X}(\downarrow, [\,], \neg))$ *is PSPACE-hard;*

(2) $\mathsf{SAT}(\mathcal{X}(\downarrow, \downarrow^*, [\,], \neg))$ *is EXPTIME-hard;*

(3) $\mathsf{SAT}(\mathcal{X}(\uparrow, [\,], =, \neg))$ *is EXPTIME-hard.*

The first two statements hold under fixed, disjunction-free DTDs.

PROOF. We show these by extending the proofs of Theorems 6.7 (for the first two) and 5.6 (for the third one), and 5.4 (for the last one).

1. $\mathsf{SAT}(\mathcal{X}(\downarrow, [\,], \neg))$ *is PSPACE-hard.* We have shown already in the proof of Theorem 6.7 that $\mathsf{SAT}(\mathcal{X}(\downarrow, [\,], \neg))$ is PSPACE-hard under fixed and disjunction-free DTDs. Indeed, observe that the fixed DTD D_0 in that proof does not use disjunction.

2. $\mathsf{SAT}(\mathcal{X}(\downarrow, \downarrow^*, [\,], \neg))$ *is EXPTIME-hard.* We modify the DTD D_1 and XPath query Q_1 used in the EXPTIME-hardness proof of $\mathsf{SAT}(\mathcal{X}(\downarrow, \downarrow^*, [\,], \neg))$ under fixed DTDs (Theorem 6.7) such that the DTD will be disjunction-free.

With negation in XPath, one can eliminate disjunctions in a DTD D as follows. Suppose that $A \to B_1 + \cdots + B_k$ is a production of D. We can replace this by $A \to B_1^*, \ldots, B_k^*$. To enforce that A has only children of a single type B_i we use the following qualifier:

$$Q_A = \neg\downarrow^*/A\left[\neg(B_1 \vee \cdots \vee B_k) \vee \bigwedge_{i\in[1,k]}\left(B_i \wedge \left(\bigvee_{j\in[1,k],i\neq j} B_j\right)\right)\right]$$

Note that we can easily rewrite Q_A into an equivalent expression without disjunction by rewriting $p_1 \vee p_2$ into $\varepsilon[\neg(\neg p_1 \wedge \neg p_2)]$. Therefore, we may assume w.l.o.g that all qualifiers introduced by eliminating disjunctions from the DTD are in $\mathcal{X}(\downarrow, \downarrow^*, [\,], \neg)$.

We now eliminate disjunctions from the DTD D_1 in the proof of Theorem 6.7 and verify that the lower bound remains valid. The new DTD D_N has the following productions:

$$\begin{array}{lll}
r \to Y_1, & Y_1 \to C, Y_2^*, L^*, & Y_2 \to C, Y_1^*, E_r^*, E_g^*, W^*, \\
W \to W^*, E_r^*, E_g^*, & L \to L^*, E_r^*, E_g^*, & E_r \to Y_1^*, W^*, L^*, \\
E_g \to \varepsilon, & C \to C^*, E_c^*, & E_c \to \varepsilon.
\end{array}$$

Along the same lines as Q_A given above, we define qualifiers Q_{Y_1}, Q_{Y_2}, Q_W, Q_L, Q_{E_r} and Q_C, specifying that the productions related to Y_1, Y_2, W, L, E_r and C are actually disjunctions. Note that in D_N we allow multiple children of the same element type and thus we have to verify whether some of the properties expressed by Q_1 in the proof of Theorem 6.7 rely on the fact that there is only a single child of a specific element type. Fortunately, all qualifiers in Q_1 check properties for *all* occurring nodes of a specific element type, in terms of universal quantifications expressed by means of negation. Hence, we can simple use the

qualifiers of Q_1 and combine them into a single $\mathcal{X}(\downarrow, \downarrow^*, [\,], \neg)$ query

$$Q = Q_1[Q_{Y_1} \wedge Q_{Y_2} \wedge Q_W \wedge Q_L \wedge Q_{E_r} \wedge Q_C].$$

It is easy to verify that (Q, D_N) is satisfiable iff Player I has a winning strategy.

3. $\mathsf{SAT}(\mathcal{X}(\uparrow, [\,], =, \neg))$ *is EXPTIME-hard.* The proof of Theorem 5.6 uses a disjunction-free DTD, and thus the same proof applies here. □

6.4. IN THE ABSENCE OF DTDS. As hinted already in Section 4, the absence of DTDs notably simplifies the analysis of $\mathsf{SAT}(\mathcal{X})$ for certain XPath classes $\mathcal{X}$.

THEOREM 6.11. *In the absence of DTDs,*

(1) *all queries in* $\mathcal{X}(\downarrow, \downarrow^*, \cup, [\,])$ *are satisfiable if label test (i.e.,* $\mathsf{lab}() = A$*) is disallowed in qualifiers; if label test is allowed,* $\mathsf{SAT}(\mathcal{X}(\downarrow, \downarrow^*, \cup, [\,]))$ *is in PTIME;*

(2) $\mathsf{SAT}(\mathcal{X}(\downarrow, \uparrow, [\,], =))$ *is in PTIME.*

PROOF. We show that in the absence of DTDs, (1) $\mathsf{SAT}(\mathcal{X}(\downarrow, \downarrow^*, \cup, [\,]))$ is in $O(|p|^3)$ time, where p is input query; furthermore, in the absence of $\mathsf{lab}() = A$, all queries in $\mathcal{X}(\downarrow, \downarrow^*, \cup, [\,])$ become satisfiable; (2) $\mathsf{SAT}(\mathcal{X}(\downarrow, \uparrow, [\,], =))$ is in $O(|p|^7)$ time.

1. $\mathsf{SAT}(\mathcal{X}(\downarrow, \downarrow^*, \cup, [\,]))$ *is in cubic time.* The proof is similar to the proof of Theorem 6.8 (1). First, observe that in the presence of label tests, not every $\mathcal{X}(\downarrow, \downarrow^*, \cup, [\,])$ query is satisfiable. For instance, $\varepsilon[\mathsf{lab}() = A \wedge \mathsf{lab}() = B]$ is not satisfiable. Thus, we need to check whether there are conflicting label tests preventing a query to be satisfiable.

To this end, we provide a decision algorithm that, given a query p in $\mathcal{X}(\downarrow, \downarrow^*, \cup, [\,])$, decides whether or not p is satisfiable in the absence of DTDs. Let *Ele* be the set of all the labels mentioned in p, plus a special symbol X not mentioned in p. As in the proof of Theorem 4.1, let $\mathcal{L}$ be the list of all the subqueries of p in "ascending" order. The algorithm uses two vectors of variables: for each subquery p' of p and each element type $A \in Ele$, (1) $\mathsf{reach}(p', A)$ collects all the element types reachable from an A element via p, and (2) $\mathsf{sat}(p', A)$ indicates the satisfiability of p' at an A element. These variables are initially set to empty set $\emptyset$ and false, respectively, and are computed based on dynamic programming. The algorithm works as follows.

(1) For each $p' \in \mathcal{L}$ (in the order of $\mathcal{L}$) and each $A \in Ele$, we compute $\mathsf{reach}(p', A)$ and $\mathsf{sat}(p', A)$, depending on the structure of p', as follows.
   (a) $p' = \varepsilon$: then $\mathsf{reach}(p', A) = \{A\}$ and $\mathsf{sat}(p', A) = \mathsf{true}$;
   (b) $p' = l$: then $\mathsf{reach}(p', A) = \{l\}$ and $\mathsf{sat}(p', A) = \mathsf{true}$;
   (c) $p' = \downarrow$: then $\mathsf{reach}(p', A) = Ele$ and $\mathsf{sat}(p', A) = \mathsf{true}$;
   (d) $p' = \downarrow^*$: then $\mathsf{reach}(p', A) = Ele$ and $\mathsf{sat}(p', A) = \mathsf{true}$;
   (e) $p' = p_1/p_2$: then $\mathsf{reach}(p', A) = \bigcup_{B \in \mathsf{reach}(p_1, A)} \mathsf{reach}(p_2, B)$ and $\mathsf{sat}(p', A) = \mathsf{true}$ if and only if $\mathsf{reach}(p', A) \neq \emptyset$;
   (f) $p' = p_1 \cup p_2$: then $\mathsf{sat}(p', A) = \mathsf{sat}(p_1, A) \vee \mathsf{sat}(p_2, A)$, and $\mathsf{reach}(p', A) = \mathsf{reach}(p_1, A) \cup \mathsf{reach}(p_2, A)$;

(g) $p' = \varepsilon[q]$: $\mathsf{sat}(p', A) = \mathsf{sat}(q, A)$, and $\mathsf{reach}(p', A) = \{A\}$ if $\mathsf{sat}(q, A) = \mathsf{true}$;
(h) $p' = [p_1]$: then $\mathsf{sat}(p', A) = \mathsf{sat}(p_1, A)$;
(i) $p' =$ '$\mathsf{lab}() = A$': then $\mathsf{sat}(p', A) = \mathsf{true}$ and $\mathsf{sat}(p', B) = \mathsf{false}$ for all other B;
(j) $p' = [q_1 \wedge \cdots \wedge q_n]$: then $\mathsf{sat}(p', A) = \mathsf{sat}([q_1], A) \wedge \cdots \wedge \mathsf{sat}([q_n], A)$;
(k) $p' = [q_1 \vee \cdots \vee q_n]$: then $\mathsf{sat}(p', A) = \mathsf{sat}([q_1], A) \vee \cdots \vee \mathsf{sat}([q_n], A)$.

(2) Return $\mathsf{sat}(p, A_1) \vee \cdots \vee \mathsf{sat}(p, A_n)$, where A_i ranges over all the labels in *Ele*.

Since $p[q] = p/\varepsilon[q]$, we can reduce the inductive case for $p[q]$ to p_1/p_2 and $\varepsilon[q]$.

The algorithm iterates over all the subqueries in $\mathcal{L}$ and all element types in *Ele*. Both of which are of size at most $O(|p|)$. Hence, the main loop in the algorithm takes at most $O(|p|^2)$ times. Each step in the loop takes at most $O(|p|)$ time, for example, the case $p' = p_1/p_2$. This brings the worst-case time complexity to $O(|p|^3)$.

We now prove the correctness of the algorithm, that is, the algorithm returns true iff p is satisfiable. Suppose that there is a tree $T \models p$. Then it is easy to verify by induction on the structure of p and the semantics of XPath queries that $\mathsf{sat}(p, r) = \mathsf{true}$, where r is the type of the root of T. More specifically, we show by induction that for any subquery p' of p and any A element n of T, if there is a B element n' such that $T \models p'(n, n')$ then (a) $B \in \mathsf{reach}(p', A)$, and (b) $\mathsf{sat}(p', A) = \mathsf{true}$; similarly for qualifier $[q]$ in p. For example, suppose that $T \models p[q](r, n)$. Then, $T \models p(r, n)$ and there exists n' in T such that $T \models q(n, n')$. Let A be the tag of n. Then, by the induction hypothesis, $A \in \mathsf{reach}(p, r)$ and $\mathsf{sat}(q, A) = \mathsf{true}$. Hence, $\mathsf{sat}(\varepsilon[q], A) = \mathsf{true}$ and by the processing of p_1/p_2 in the algorithm, we also have that $\mathsf{sat}(p, r) = \mathsf{true}$. Other cases can be verified similarly.

Conversely, if the algorithm returns true, we define an XML tree $\mathsf{Tree}(p)$ that satisfies p. We construct $\mathsf{Tree}(p)$ top-down, by induction on the structure of p. Initially $\mathsf{Tree}(p)$ consists of a single root v, which has the label r where $\mathsf{sat}(p, r) = \mathsf{true}$ (if there are multiple labels r's such that $\mathsf{sat}(p, r) = \mathsf{true}$, we randomly pick one). For each node u, we maintain its label, referred to as $\mathsf{lab}(u)$, and a subquery of p, referred to as $\mathsf{query}(u)$, which the subtree of u should satisfy. For example, for the root v, $\mathsf{lab}(v) = r$ and $\mathsf{query}(v) = p$. Upon the completion of the subtree of u we find a node u' in the subtree, referred to as $E(u, \mathsf{query}(u))$, such that $\mathsf{lab}(u') \in \mathsf{reach}(\mathsf{query}(u), \mathsf{lab}(u))$. We generate $\mathsf{Tree}(p)$ by repeatedly selecting a node u and construct the subtree of u based on $\mathsf{query}(u)$, as follows: (a) If $\mathsf{query}(u) = \varepsilon$, then nothing needs to be done. (b) If $\mathsf{query}(u) = A$, then we generate an A element u' as the child of u, and set $\mathsf{lab}(u') = A$, $\mathsf{query}(u') = \varepsilon$ and $E(u, \mathsf{query}(u)) = u'$. (c) If $\mathsf{query}(u)$ is $\downarrow$ or $\downarrow^*$, we generate an X element u' as the child of u, where X is label variable whose value is to be fixed later. We set $\mathsf{lab}(u') = X$, $\mathsf{query}(u') = \varepsilon$ and $E(u, \mathsf{query}(u)) = u'$. (d) If $\mathsf{query}(u) = p_1/p_2$, then we first construct $\mathsf{Tree}(p_1)$ rooted at u and then generate $\mathsf{Tree}(p_2)$ rooted at $u' = E(u, p_1)$. This is possible since $\mathsf{sat}(p_1, \mathsf{lab}(u)) = \mathsf{true}$ and there exists $B \in \mathsf{reach}(p_1, \mathsf{lab}(u))$ such that $\mathsf{sat}(p_2, B) = \mathsf{true}$. We set $E(u, p_1/p_2) = E(u', p_2)$. (e) If $\mathsf{query}(u) = p_1 \cup p_2$, then $\mathsf{Tree}(p)$ is $\mathsf{Tree}(p_1)$ and we set $E(u, p_1 \cup p_2) = E(u, p_1)$ if $\mathsf{sat}(p_1, \mathsf{lab}(u)) = \mathsf{true}$; otherwise, we must have that $\mathsf{sat}(p_2, \mathsf{lab}(u)) = \mathsf{true}$ and thus we set $\mathsf{Tree}(p)$ to be $\mathsf{Tree}(p_2)$

and $E(u, p_1 \cup p_2) = E(u, p_2)$. (f) If $\mathsf{query}(u) = \varepsilon[q]$, then $\mathsf{Tree}(p)$ is $\mathsf{Tree}(q)$. (g) If $\mathsf{query}(u) = [p_1]$, then $\mathsf{Tree}(p)$ is $\mathsf{Tree}(p_1)$. (i) If $\mathsf{query}(u)$='$\mathsf{lab}() = A$', we consider two cases. If $\mathsf{lab}(u)$ is a variable X, then we set X to be A. Otherwise, $\mathsf{lab}(u)$ is a label in *Ele*; by the construction, we have that $\mathsf{sat}(\mathsf{query}(u), \mathsf{lab}(u)) = \mathsf{true}$; thus, we must have that $\mathsf{lab}(u) = A$; in this case, nothing needs to be done. (j) If $\mathsf{query}(u) = [q_1 \wedge \cdots \wedge q_n]$ then $\mathsf{sat}(q_i, \mathsf{lab}(u)) = \mathsf{true}$ for all $i \in [1, n]$. We construct $\mathsf{Tree}(p)$ by adding each $\mathsf{Tree}(q_i)$ as a subtree of u. Note that we can freely generate a separate branch for each qualifier q_i because of the existential semantics of $\mathcal{X}(\downarrow, \downarrow^*, \cup, [\,])$ and the absence of DTDs. This is no longer the case in the presence of DTDs. (k) If $\mathsf{query}(u) = [q_1 \vee \cdots \vee q_n]$, then there must be some q_i such that $\mathsf{sat}(q_i, \mathsf{lab}(u)) = \mathsf{true}$. We construct $\mathsf{Tree}(p)$ by adding the corresponding $\mathsf{Tree}(q_i)$ as a subtree of u. The process proceeds until $E(r, p)$ is found. At this point, for any label variable X that has not yet been instantiated, we set X to be an arbitrary label in *Ele*; indeed, one can verify that in this case the value of X has no impact on $\mathsf{sat}(p, r)$. Given that $\mathsf{sat}(p, r) = \mathsf{true}$, there are no conflicting label tests at any node in the construction of $\mathsf{Tree}(p)$, and by the semantics of XPath, it is easy to verify that $\mathsf{Tree}(p) \models p$.

Observe that in the absence of label tests, queries in $\mathcal{X}(\downarrow, \downarrow^*, \cup, [\,])$ are always satisfiable since $\mathsf{sat}(p, A)$ is always true for any $A \in \mathit{Ele}$, as shown by the algorithm and proof above.

2. $\mathsf{SAT}(\mathcal{X}(\downarrow, \uparrow, [\,], =))$ *is in time* $O(|p|^7)$. We reduce $\mathsf{SAT}(\mathcal{X}(\downarrow, \uparrow, [\,], =))$ to the satisfiability problem for conjunctive queries on XML trees. Given a query p in $\mathcal{X}(\downarrow, \uparrow, [\,], =)$, let Σ_{doc} be the signature containing predicates specifying the query p: (1) unary predicates $P_a(x)$ for each tag a; (2) unary predicate $\mathit{Root}(x)$; (3) binary predicate $R_{child}(x, y)$; (4) relations $R_{a,b,\mathsf{op}}(x, y)$ for each pair of attributes a, b and each operator op; and (5) relations $R_{a,c,\mathsf{op}}(x)$ for each attribute a, constant value c and operator op in p.

For each XML tree T there is a corresponding structure for Σ_{doc}, where the elements of the structure are the nodes of T, the predicate Root holds only at the root; P_a holds for exactly those nodes labeled with a; the predicate $R_{child}(n_1, n_2)$ holds iff n_2 is a child of n_1; and $R_{a,b,\mathsf{op}}(n_1, n_2)$ holds iff $n_1.a$ op $n_2.b$, and similarly $R_{a,c,\mathsf{op}}(n_1)$ holds iff $n_1.a$ op c.

Let CQD be the set of conjunctive queries over Σ_{doc} atomic formulas. Such queries are evaluated using the standard semantics of first-order logic.

The fact below follows directly from the XPath semantics given in Section 2, which can be easily formalized as a linear-time translation into first-order logic. One can verify that disjunction is required only when union and disjunction are present in the XPath expression, and similarly negation is generated only via negation in the expression; hence, the queries resulting from $\mathcal{X}(\downarrow, \uparrow, [\,], =)$ will be in CQD.

LEMMA 6.12. *There is a linear time function converting each* $\mathcal{X}(\downarrow, \uparrow, [\,], =)$ *expression into an equivalent* CQD *query.*

We now claim that satisfiability of CQD queries over XML trees is in time $O(|Q|^7)$. Our algorithm will use a variant of the well-known technique of canonical databases. Given an input CQD query Q, let E be the smallest equivalence relation on variables such that:

—$E(x, y)$ holds when $x = y$ is a conjunct of Q;
—$E(x, x')$ holds if $E(y, y')$ holds and $R_{child}(x', y')$, $R_{child}(x, y)$ are conjuncts of Q;
—$E(x, x')$ holds if $Root(x)$ and $Root(x')$ are both conjuncts of Q.

Note that E can be calculated in quadratic time from Q.

Let F be the set consisting of all pairs (x, b), where x is a variable and b is a constant, unioned with the set c of constant symbols mentioned in Q. Let E_2 be the smallest equivalence relation on F containing:

—$E_2((x, a), (y, b))$, where $R_{a,b,=}(x, y)$ is a conjunct of Q;
—$E_2((x, a), (x', a))$, where $E(x, x')$ holds;
—$E_2((x, a), c)$, where c is a constant and $R_{b,c,=}(y)$ is a conjunct of Q.

Note that E_2 is the transitive closure of a relation on F of size at most $|Q|^3$; the first and third items above contribute at most one edge for each conjunct of Q while the second contributes at most one edge for each attribute in Q and each edge in E. Hence, E_2 can be calculated in at worst $|Q|^6$ time.

We say Q is *cogent* if:

—If $E_2((x, a), (y, b))$ holds, then $R_{a,b,\neq}(x, y)$ is not a conjunct of Q, and similarly for $E_2((x, a), c)$.
—If $E(x, x')$ and $a(x)$ are conjuncts of Q, then no $b(x)$ is a conjunct of Q for $b \neq a$.
—If $E(x, x')$ holds and $Root(x)$ is a conjunct of Q, then there is no $R_{child}(y, x')$ in Q.

Suppose that Q is cogent. Let Rep_E be the set of equivalence classes of E and Rep_2 be the set of equivalence classes of E_2. We define the structure $\mathsf{CM}(Q)$ as follows:

—The domain of $\mathsf{CM}(Q)$ consists of one node for each element of Rep_E.
—For each $e \in \mathsf{Rep}_E$, the label of e is set to a if $a(y)$ is a conjunct of Q for some $y \in e$, and is set arbitrarily otherwise.
—For $e, e' \in \mathsf{Rep}_E$, e is a child of e' iff for some $x \in e$, $y \in e'$, $R_{child}(y, x)$ is in Q.
—$x.a = f$ for $f \in \mathsf{Rep}_2$ if $(x, a) \in f$.

We will verify below that the above requirements give a well-defined XML tree if Q is cogent. We now claim:

LEMMA 6.13. *Q is satisfiable by an XML tree iff Q is cogent and the child relation of $\mathsf{CM}(Q)$ is acyclic.*

If the lemma holds, then satisfiability of Q can be checked in $O(|Q|^7)$ time. Indeed, the first requirement in the definition of cogent can be checked in time $O(|Q|^7)$, since we can calculate a representation of E_2 in time $|Q|^6$ and then check each conjunct of Q by iterating through this representation. The second requirement requires iterating through pairs of conjuncts in Q and checking them against elements in E; since E has size quadratic in $|Q|$, this can be done in cubic time. The last requirement requires iterating over pairs in E, which can be done in quadratic time. Furthermore, cyclicity is checkable in quadratic time. Note that in

the case of XPath, the cyclicity condition is redundant (since conjunctive queries generated from XPath can be taken to be acyclic).

We next prove Lemma 6.13. We first argue that if Q is satisfiable then Q is cogent and $\mathsf{CM}(Q)$ is acyclic. If Q is satisfiable then there is a homomorphism h from the variables of Q to nodes in some XML tree T; homomorphism here means that for a conjunct $\rho(x, y)$ of Q, $\rho(x, y)$ holds iff $\rho(h(x), h(y))$ holds. If we fix such an h, then we can see from the definition of homomorphism that:

—$E(x, y)$ implies $h(x) = h(y)$;
—$E_2((x, a), (y, b))$ implies $h(x).a = h(y).b$; and
—$E_2((x, a), c)$ implies $h(x).a = c$.

The first part of the definition of cogent follows from the first item above, since if $x \neq y$ were a conjunct of Q we would have $h(x) \neq h(y)$. The rest of the definition of cogent follows similarly from the second and third item above.

If $\mathsf{CM}(Q)$ were cyclic, we would have a sequence of variables x_i with $R_{child}(x_i, x_{i+1})$ as conjuncts of Q. Then, if h is the homomorphism above, we would have $R_{child}(h(x_i), h(x_i))$, violating that the satisfying model is a tree.

We now turn to the other direction, showing that if Q is cogent and $\mathsf{CM}(Q)$ is acyclic then Q is satisfiable by an XML tree. We verify that $\mathsf{CM}(Q)$ is well defined if Q is cogent. Since Q is cogent, every node in $\mathsf{CM}(Q)$ has exactly one label; hence, the labeling function is well defined. If e is a node of $\mathsf{CM}(Q)$, we need also to show that the value of attribute a on e is well defined. By the definition above, this is the case as long as for every x, x' in the equivalence class e, $E_2(x.a, x'.a)$. From the definition of E_2, it follows that $E(x, x')$ implies $E_2(x'.a, x.a)$. We now verify that $\mathsf{CM}(Q)$ is a forest. Suppose a node $e \in \mathsf{CM}(Q)$ has more than one immediate predecessor. Then, we have $R_{child}(y, x)$ and $R_{child}(y', x')$ where $E(x, x')$ but not $E(y, y')$. But this violates the last component in the definition of E. The fact that there are no cycles in the child relation of $\mathsf{CM}(Q)$ is true by assumption. Note that $\mathsf{CM}(Q)$ may not be a tree, since there may be several nodes with no predecessor. If there is no conjunct of the form $Root(x)$ in Q, then we extend $\mathsf{CM}(Q)$ to a tree by adding a root node with an arbitrary label. If there is a conjunct of the form $Root(x)$, then modify $\mathsf{CM}(Q)$ as follows. Let c_e be the connected component of e in $\mathsf{CM}(Q)$. Since Q is cogent, e must be the root of this component (since otherwise the third condition in cogent is violated). For every other node e' in $\mathsf{CM}(Q)$ that has no predecessor, attach this node as a child of one of the leaves of c_e. In either case, we call the resulting structure $\mathsf{CM}'(Q)$.

We now show that $\mathsf{CM}'(Q)$ satisfies Q. Let h be the mapping taking a variable x to its E equivalence class. We claim that this mapping witnesses the satisfiability of $\mathsf{CM}(Q)$ as required. By the definition of $\mathsf{CM}(Q)$ any label equalities or parent/child relations asserted of variables in Q hold on the images of the variables under h. If $x.a = y.b$ is asserted in Q, then the first part of the definition of E_2 implies that $E_2((x, a), (y, b))$ holds, so $h(x).a = h(y).b$ in $\mathsf{CM}(Q)$, hence in $\mathsf{CM}'(Q)$. Similarly the last part of the definition of E_2 implies that relations $x.a = c$ are preserved.

Suppose $x.a \neq y.b$ is in Q. Since Q is cogent, we know that $((x, a), (y, b))$ is not in E_2. Hence $h(x).a \neq h(y).b$. Similarly, the fact that Q is cogent implies that conjuncts of Q of the form $x.a \neq c$ are satisfied in $\mathsf{CM}(Q)$, hence in $\mathsf{CM}'(Q)$ as well. Finally, if $Root(x)$ is in Q, then the construction of $\mathsf{CM}'(Q)$ guarantees that the E-equivalence class of x is the root of $\mathsf{CM}'(Q)$. □

However, as with the disjunction-free DTDs, in the presence of data value equality ($=$) or upward modalities ($\uparrow$), the lack of a DTD does not help matters.

COROLLARY 6.14. *In the absence of DTDs, the following problems are NP-hard:* (1) $\mathsf{SAT}(\mathcal{X}(\cup, [\,], =))$, *and* (2) $\mathsf{SAT}(\mathcal{X}(\downarrow, \uparrow, \cup, [\,]))$.

PROOF. The proofs follow from that of Theorem 6.9. Its last part also follows from the results of Hidders [2004].

1. $\mathsf{SAT}(\mathcal{X}(\cup, [\,], =))$ *is NP-hard.* The proof is the same as the proof of case (1) of Theorem 6.9. Indeed, the fixed DTD D_0 used in that proof can be omitted since the qualifiers in that proof encode the 3SAT instance completely.

2. $\mathsf{SAT}\mathcal{X}(\uparrow, \cup, [\,])$ *is NP-hard.* Again the proof is the same as the proof of Theorem 6.9 (3). Indeed, we can omit the fixed DTD D_2 in that proof, since the existence of the desired chain of T and F nodes is already encoded entirely by the qualifier in that proof. □

As in the settings of disjunction-free and fixed DTDs, the absence of DTDs does not simplify the satisfiability analysis of fragments with negation, as shown below.

COROLLARY 6.15. *In the absence of DTDs,*

(1) $\mathsf{SAT}(\mathcal{X}(\downarrow, [\,], \neg))$ *is PSPACE-hard;*
(2) $\mathsf{SAT}(\mathcal{X}(\downarrow, \downarrow^*, [\,], \neg))$ *is EXPTIME-hard;*
(3) $\mathsf{SAT}(\mathcal{X}(\uparrow, [\,], =, \neg))$ *is EXPTIME-hard.*

PROOF. The proofs are mild extensions of the proofs of Theorem 6.7(1), Corollary 6.10(2), and Theorem 5.6, respectively.

1. $\mathsf{SAT}(\mathcal{X}(\downarrow, [\,], \neg))$ *is PSPACE-hard.* We encode a minor modification of the DTD D_0 used in the proof of Theorem 6.7(1), in terms of the following qualifiers. Let $\phi = Q_1 x_1 Q_2 x_2 \cdots Q_m x_m \phi$ be an instance of Q3SAT.

(1) $r \to X$. To ensure that the root has an X child, we use $q_r = [X]$. Note that this actually encodes $r \to X^*$.
(2) $X \to T^*, F^*$. If $Q_i = \forall$, the qualifier q_i defined in the proof of Theorem 6.7(1) already ensures that X has both T and F children. However, if $Q_i = \exists$, q_i does not assure that there exist either T or F children. Therefore, in this case, we change q_i to $q_i = \neg\downarrow^{2(i-1)}/X[\neg T \wedge \neg F \wedge \neg(T \wedge F)]$.
(3) $T \to X$ and $F \to X$. To ensure that up to the depth that we are interested in, each $\downarrow^{(2i-1)}/T$ and $\downarrow^{(2i-1)}/F$ has an X child, we define the qualifiers

$$qt_i = \neg\downarrow^{2i-1} T[\neg X], \quad \text{and} \quad qf_i = \neg\downarrow^{2i-1} F[\neg X].$$

Note again that this actually encodes $T \to X^*$ and $F \to X^*$.

We remark that it is harmless to encode a modification of the DTD D by introducing Kleene stars, since the reduction proof does not rely on cardinality constraints.

Finally, we define the following query in $\mathcal{X}(\downarrow, [\,], \neg)$:

$$\mathsf{XP}(\phi) = \varepsilon\left[\bigwedge_{j\in[1,n]} \neg\mathsf{XP}(C_j) \wedge q_r \wedge \bigwedge_{i\in[1,m]} (q_i \wedge qt_i \wedge qf_i)\right],$$

where the $\mathsf{XP}(C_j)$'s are given in the proof of Theorem6.7(1). One can verify that $\mathsf{XP}(\phi)$ is satisfiable iff ϕ is satisfiable.

2. $\mathsf{SAT}(\mathcal{X}(\downarrow, \downarrow^*, [\,], \neg))$ *is EXPTIME-hard.* We know that $\mathsf{SAT}(\mathcal{X}(\downarrow, \downarrow^*, [\,], \neg))$ is EXPTIME-hard under fixed, disjunction-free DTDs (Corollary 6.10(2)). We now show that the DTD D_N used in the proof of Corollary 6.10 (2) can be encoded using qualifiers. Observe that qualifiers Q_{Y_1}, Q_{Y_2}, Q_W, Q_L, Q_{E_r} and Q_C already encode the corresponding productions in D_N. Thus we only need to encode the productions $r \to Y_1, E_g \to \varepsilon$ and $E_c \to \varepsilon$. These can be expressed by $Q_r = r[Y_1]$, $Q_{E_g} = \neg\downarrow^*/E_g[\downarrow]$, and $Q_{E_c} = \neg\downarrow^*/E_c[\downarrow]$, respectively. Recall the query Q from the proof of Corollary 6.10(2). Let $Q' = Q[Q_r \wedge Q_{E_g} \wedge Q_{E_c}]$. Then, it can be easily verified that Q' is satisfiable if and only iff Player I has a winning strategy.

3. $\mathsf{SAT}(\mathcal{X}(\uparrow, [\,], =, \neg))$ *is EXPTIME-hard.* We prove the EXPTIME-hardness by extending the proof of Theorem 5.6. Recall the query Q and the DTD D_0 used in that proof. Note that D_0 defines XML trees consisting of a root with a list of C children. We define a query Q_1, which extends Q by adding a conjunct to the set of qualifiers of Q at the root:

$$Q_{att} = \neg C\left[\neg\left(\varepsilon/@h \;\wedge\; \varepsilon/@k \;\wedge\; \varepsilon/@\mathsf{next} \;\wedge \bigwedge_{i\in[1,n]} \varepsilon/@t_i\right)\right].$$

Here, Q_{att} asserts that every C element has all the qualifiers specified in D_0. With this addition, the instance of TPG-CT is entirely encoded by the query Q_1, and the DTD D_0 is no longer needed. The rest of the proof is the same as its counterpart for Theorem 5.6. □

7. *XPath Queries with Sibling Axes*

Since XML data is ordered, it is often desirable to access this order using XPath sibling axes. In this section we revisit the satisfiability problem for XPath fragments with the sibling axes. Our aim is to explore the differences and similarities between horizontal modalities and their vertical counterparts, and to investigate the impact of the sibling axes on the satisfiability analysis. As will be seen shortly, in many cases the presence of sibling axes complicates the satisfiability analysis.

7.1. ADDING SIBLING AXES TO XPATH. We extend the definition of XPath queries given in Section 2 by including sibling axes:

$$p \quad ::= \quad \ldots \;\mid\; \rightarrow \;\mid\; \rightarrow^* \;\mid\; \leftarrow \;\mid\; \leftarrow^*.$$

The semantics of these operators is given below, as an extension of the interpretation of $T \models p(n, n')$ presented in Section 2:

—if $p = \rightarrow$, then n' is the immediate right sibling of n;
—if $p = \rightarrow^*$, then n' is either n or a right sibling of n;
—if $p = \leftarrow$, then n' is the immediate left sibling of n;
—if $p = \leftarrow^*$, then n' is either n or a left sibling of n.

Observe that '$\rightarrow^+$' (i.e., $\rightarrow^* \cup \rightarrow$; respectively, '$\leftarrow^+$') is the following-sibling (respectively, preceding-sibling) axis of XPath 1.0, and '$\rightarrow$' (respectively, '$\leftarrow$')

denotes the immediate right sibling (respectively, the immediate left sibling). Although XPath 1.0 does not explicitly define '$\leftarrow$, $\rightarrow$', these operators are definable in terms of the preceding-sibling and following-sibling axes, together with *position()*:

$$\leftarrow \;=\; \leftarrow^{+}[\mathit{position}() = 1], \qquad\qquad \rightarrow \;=\; \rightarrow^{+}[\mathit{position}() = 1].$$

7.2. SATISFIABILITY ANALYSIS. First, consider XPath fragments without qualifiers. Recall that $\mathsf{SAT}(\mathcal{X}(\downarrow, \uparrow))$ is NP-hard (Proposition 4.3). In contrast, we show that sibling axes simplify the satisfiability analysis. Indeed, if we substitute $\rightarrow$ and $\leftarrow$ for $\downarrow$ and $\uparrow$ in $\mathcal{X}(\downarrow, \uparrow)$, the satisfiability analysis becomes tractable. The difference between $\mathcal{X}(\downarrow, \uparrow)$ and $\mathcal{X}(\rightarrow, \leftarrow)$ is that while a query in $\mathcal{X}(\downarrow, \uparrow)$ can constrain the subtree of a node by moving downward and upward repeatedly in the *subtree*, queries in $\mathcal{X}(\rightarrow, \leftarrow)$ are not able to do it: as soon as the navigation moves down in a tree, it cannot move back to the same node.

THEOREM 7.1. $\mathsf{SAT}(\mathcal{X}(\rightarrow, \leftarrow))$ *is in PTIME in the presence of DTDs.*

PROOF. Similar to the proof of Theorem 4.1, we provide a decision algorithm based on dynamic programming. Given a query p in $\mathcal{X}(\rightarrow, \leftarrow)$ and a DTD $D = (\mathit{Ele}, \mathit{Att}, P, R, r)$, the algorithm decides whether or not (p, D) is satisfiable in PTIME.

Observe that any query $p \in \mathcal{X}(\rightarrow, \leftarrow)$ is of the form $A_1/\eta_1/\cdots/A_n/\eta_n$, where for each $i \in [1, n]$, A_i is an element type representing a one-step downward move, and η_i is a sequence of $\rightarrow$ and $\leftarrow$ indicating multi-step horizontal moves. In other words, on an XML tree T, the navigation of p at the i-th step first moves sideways, and then downward; moreover, as soon as it moves downward, the navigation proceeds to lower levels of T without looking back upward. Note that a query in $\mathcal{X}(\rightarrow, \leftarrow)$ is not satisfiable if it starts with $\rightarrow$ or $\leftarrow$. Let p_i denote $A_i/\eta_i/\cdots/A_n/\eta_n$; note that p_1 is p.

We now define the variables to be used in the algorithm. (a) For each p_i and each $A \in \mathit{Ele}$, we define a Boolean variable $\mathsf{sat}(p_i, A)$ that indicates whether or not p_i is satisfiable at an A element. (b) For each $A \in \mathit{Ele}$, let M_A be an NFA representing the regular expression $P(A)$, such that each state of M_A is reachable from the start state, and there exists no ε-transition. Such an M_A can be computed in PTIME, and the size $|M_A|$ of the NFA is linear in the size $|P(A)|$. For each η_i and each $B \in \mathit{Ele}$, let $\mathsf{reach}(M_A, B, \eta_i)$ be the set consisting of element types C such that there exist two states q_1, q_2 in M_A and (i) there exists an outgoing transition from q_1 labeled B; and (ii) q_2 can be reached from q_1 via B/η_1 ($\rightarrow$ for forward move and $\leftarrow$ for backward); and (iii) the last transition is via an edge labeled C. It is easy to verify that $\mathsf{reach}(M_A, B, \eta_i)$ can be computed in PTIME by, for example, representing M_A as a graph with inverse edges (for $\leftarrow$), and computing $\mathsf{reach}(M_A, B, \eta_i)$ based on dynamic programming.

Using these variables, the decision algorithm works as follows: It first computes $\mathsf{reach}(M_A, B, \eta_i)$ for each $i \in [1, n]$ and all $A, B \in \mathit{Ele}$. Then, for each p_i in the decreasing order (i.e., for $i = n, n-1, \ldots, 1$), and for each $A \in \mathit{Ele}$, it computes $\mathsf{sat}(p_i, A)$, which is true iff (a) $\mathsf{reach}(M_A, A_i, \eta_i)$ is not empty; (b) there exists $B \in \mathsf{reach}(M_A, A_i, \eta_i)$ such that $\mathsf{sat}(p_{i+1}, B)$ is true. Both steps can be done in PTIME in $|D|$ and $|p|$. Hence, the algorithm is in PTIME. □

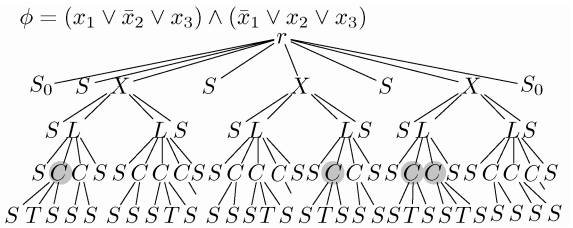

FIG. 9. A tree of the DTD encoding a 3SAT instance in the proof of Theorem 7.2. The grey discs indicate which clauses are satisfied. For simplicity we omitted the children of each 3rd C node.

Once qualifiers are added to positive XPath fragments with sibling axes, we get again into the realm of intractability:

PROPOSITION 7.2. $\mathsf{SAT}(\mathcal{X}(\rightarrow, [\ ]))$ *is NP-hard under fixed, disjunction-free and nonrecursive DTDs.*

PROOF. We show the intractability by reduction from 3SAT. Consider an instance $\phi = C_1 \wedge \cdots \wedge C_n$ of 3SAT, and assume that the variables in ϕ are $x_1, \ldots, x_m$. We first define a fixed, disjunction-free and nonrecursive DTD $D = (Ele, Att, P, R, r)$:

$Ele = \{r, S, S_0, X, L, C, T\}$.
P: $r \rightarrow S_0, (S, X)^*, S_0$, $\quad X \rightarrow S, L, L, S$, $\quad L \rightarrow S, C^*, S$,
$C \rightarrow S, T^*, S$, $\quad S_0 \rightarrow \varepsilon$, $\quad S \rightarrow \varepsilon$, $\quad T \rightarrow \varepsilon$.
$Att = \emptyset$, $R(A) = \emptyset$ for all $A \in Ele$.

An XML tree T conforming to D_2 is depicted in Figure 9. The root of T consists of a list of X elements. As will been seen shortly, these X elements encode variables $x_1, \ldots, x_m$. Below each X element there are two L elements, where the first L element encodes the case when the corresponding variable is true, and the second L element encodes false. Below each L is a list of C elements, while each C element may have a T child. The elements S, S_0 serve as delimiters to indicate the start and end of children lists, respectively.

We encode ϕ in terms of the DTD D and a query $\mathsf{XP}(\phi)$ in $\mathcal{X}(\rightarrow, [\ ])$, defined with the following qualifiers at the root.

(1) *Variables*: $q_v = S_0/{\rightarrow}^{2m}/{\rightarrow}[\mathsf{lab}() = S_0]$, where ${\rightarrow}^{2m}$ is a shorthand for the $2m$-fold concatenation of $\rightarrow$. This asserts that the (S, X) list under r contains precisely m elements of type X. We use X_j to denote $S_0/{\rightarrow}^{2j}$, which encodes the variable x_j in ϕ.

(2) *The connection between clauses and literals*:

$$q_c = \bigwedge_{i \in [1,n],\, j \in [1,m]} (q^T_{i,j} \wedge q^F_{i,j}),$$

where

$$q^T_{i,j} = \begin{cases} X_j/S/{\rightarrow}/S/{\rightarrow}^i/S/{\rightarrow}[\mathsf{lab}() = T] & \text{if } x_j \text{ appears in } C_i \\ X_j/S/{\rightarrow}/S/{\rightarrow}^i/S/{\rightarrow}[\mathsf{lab}() = S] & \text{otherwise} \end{cases}$$

$$q^F_{i,j} = \begin{cases} X_j/S/{\rightarrow}/{\rightarrow}/S/{\rightarrow}^i/S/{\rightarrow}[\mathsf{lab}() = T] & \text{if } \bar{x}_j \text{ appears in } C_i \\ X_j/S/{\rightarrow}/{\rightarrow}/S/{\rightarrow}^i/S/{\rightarrow}[\mathsf{lab}() = S] & \text{otherwise} \end{cases}$$

We encode the clause C_i in terms of C^i under both the first and second L child of X, where C^i is a shorthand for $S/{\rightarrow}^i$, that is, the ith C child of L from the left. For each variable x_j (i.e., X_j), if x_j appears (positively) in C_i, then $q^T_{i,j}$ ensures that C^i under the first L has a T child, that is, C_i is satisfied if x_j is true; otherwise C^i under the first L has no T child; similarly, $q^F_{i,j}$ encodes the connection between $\bar{x}_j$ (i.e., x_j appears negatively) and C_i. Note that these also assert that below each L there are at least n many C children.

(3) *Truth assignment*:

$$q_a = \bigwedge_{j\in[1,m]} (X_j[L/S/{\rightarrow}^{n+1}[\mathsf{lab}() = S] \ \wedge \ L/S/{\rightarrow}^{n+2}[\mathsf{lab}() = S]).$$

We encode x_j (i.e., X_j) such that it is assigned true if under the first L child of X_j there are precisely n elements of C type; similarly, x_j is false if under the second L child of X_j there are n elements of C type. The qualifier q_a asserts that for each x_j there is a single truth value, only one of the C lists under X^j has n elements of type C.

(4) *Clauses*:

$$q_\phi = \bigwedge_{i\in[1,n]} X/L[S/{\rightarrow}^{n+1}[lab() = S]]/C^i[T].$$

This asserts that all clauses C_i (e.g., C^i) must be satisfied for the truth assignment.

Taken together, the query $\mathsf{XP}(\phi)$ is defined to be $\varepsilon[q_v \wedge q_c \wedge q_a \wedge q_\phi]$. One can easily verify that $\mathsf{XP}(\phi)$ is satisfiable by an XML tree of the fixed, disjunction-free and nonrecursive DTD D iff ϕ is satisfiable.

A mild variation of the proof above suffices to show that $\mathsf{SAT}(\mathcal{X}(\leftarrow, [\ ]))$ is NP-hard. □

7.3. Satisfiability with Sibling Axes and Negation. When negation is introduced together with sibling axes, the situation is as bad as without sibling axis, and may be worse. Indeed, similar to the result that $\mathsf{SAT}(\mathcal{X}(\downarrow, [\ ], \neg))$ is PSPACE-hard (Theorem 5.2), we show that the lower bound remains intact if we substitute $\rightarrow$ (respectively, $\leftarrow$) for $\downarrow$ in the fragment, even when the DTDs are restricted or absent. Below, by a *no-star DTD*, we mean that none of the productions in the DTD contains the Kleene star.

Proposition 7.3. $\mathsf{SAT}(\mathcal{X}(\rightarrow, [\ ], \neg))$ *is PSPACE-hard either* (1) *under nonrecursive and no-star DTDs; or* (2) *in the absence of DTDs.*

Proof. We prove these by reduction from Q3SAT. Consider an instance $\phi = Q_1x_1Q_2x_2\cdots Q_mx_m\ E$ of Q3SAT, as described in the proof of Proposition 5.1.

1. Under nonrecursive and no-star DTDs. Given an instance ϕ of Q3SAT, we first define a no-star and nonrecursive DTD $D = (Ele, Att, P, R, r)$ as follows:

$Ele = \{X, T, F, S\}$.
P: $r \rightarrow S, X, \ldots, X$ /* n occurrences of X */ $X \rightarrow S, (T + \varepsilon), (F + \varepsilon)$,
$T \rightarrow \varepsilon$, $F \rightarrow \varepsilon$, $S \rightarrow \varepsilon$.
$Att = \emptyset$, $R(A) = \emptyset$ for all $A \in Ele$.

Intuitively, we use the ith X child of the root is to encode the variable x_i, which has a truth assignment T or F.

We next encode $\phi = \varepsilon[q_1 \wedge q_2]$ in terms of a query $\mathsf{XP}(\phi)$ in $\mathcal{X}(\rightarrow, [\ ], \neg)$, where q_1 encodes the quantifiers $Q_1 x_1 Q_2 x_2 \cdots Q_m x_m$ in ϕ, and q_2 encodes the the 3SAT instance $E = C_1 \wedge \cdots \wedge C_n$. More specifically, $q_1 = \bigwedge_{i \in [1,m]} q^i$, where $q^i = S/{\rightarrow}^i[S/\rightarrow/\rightarrow]$ if Q_i is '$\forall$', and $q^i = S/{\rightarrow}^i[S/\rightarrow[\neg\rightarrow]]$ if Q_i is '$\exists$'; here ${\rightarrow}^i$ denotes the i-fold concatenation of $\rightarrow$. That is, if x_i is universally quantified, then both T and F values of x_i should be considered; otherwise only one of these truth values is needed. We define q_2 to be $(\neg c_1 \wedge \cdots \wedge \neg c_n)$, where c_i represents the *negation* of the clause C_i. More specifically, let $C_i = l_i^1 \vee l_i^2 \vee l_i^3$, where l_i^j is a literal, that is, it is either a variable x_i or the negation $\bar{x}_i$ of a variable. Then, c_i is to code $\neg l_i^1 \wedge \neg l_i^2 \wedge \neg l_i^3$. Without loss of generality, assume that the variables of these literals are x_s, x_t and x_u, respectively, with $s < t < u$. Then, c_i is defined as $S/{\rightarrow}^s[Z_s]/{\rightarrow}^{t-s}[Z_t]/{\rightarrow}^{u-t}[Z_u]$, where $Z_j = F_j$ if x_j appears in c_i, and $Z_j = T_j$ if $\bar{x}_j$ appears in c_i, for j ranging over s, t, u. For example, if $C_i = x_s \vee \bar{x}_t \vee x_u$ with $s < t < u$, then $c_i = S/{\rightarrow}^s[F]/{\rightarrow}^{t-s}[T]/{\rightarrow}^{u-t}[F]$. Intuitively, $\neg c_i$ is to assert that for all possible truth assignments consistent with q_1, the clause C_i is true. One can verify that $(\mathsf{XP}(\phi), D)$ is satisfiable if and only if ϕ is true.

2. In the absence of DTDs. It suffices to show that the DTD D can be encoded in terms of qualifiers in $\mathcal{X}(\rightarrow, [\ ], \neg)$ along the same lines as the proof of Theorem 6.15(3). For the productions $r \rightarrow S, X, \ldots, X$ and $X \rightarrow S, (T + \varepsilon), (F + \varepsilon)$, we define Q_r and Q_X as:

$$Q_r = S\left[\bigwedge_{i \in [1,n]} {\rightarrow}^i[\mathsf{lab}() = X]] \wedge \neg S[{\rightarrow}^{n+1}\right],$$

$$\begin{aligned} Q_X = {} & \neg X[\neg S] \wedge \neg X/S[{\rightarrow}^3] \wedge \neg X/S[\rightarrow \wedge \neg{\rightarrow}^2 \wedge \rightarrow[\mathsf{lab}() \neq T \wedge \mathsf{lab}() \neq F]] \\ & \neg X/S[{\rightarrow}^2 \wedge \neg(\rightarrow[\mathsf{lab}() = T] \wedge {\rightarrow}^2[\mathsf{lab}() = F])]. \end{aligned}$$

Let $Q = \mathsf{XP}(\phi)[Q_r \wedge Q_X]$, where $\mathsf{XP}(\phi)$ is the query in the proof of part (1) above. Then, one can verify that Q is satisfiable iff $(\mathsf{XP}(\phi), D)$ is satisfiable. Thus, from the proof of part (1), it follows that the PSPACE-hardness result holds in the absence of DTDs.

Similarly, we can show that $\mathsf{SAT}(\mathcal{X}(\leftarrow, [\ ], \neg))$ is PSPACE-hard in these settings. □

THEOREM 7.4. $\mathsf{SAT}(\mathcal{X}(\downarrow, \uparrow, \leftarrow, \rightarrow, \leftarrow^*, \rightarrow^*, \cup, [\quad], \neg))$ *is PSPACE-complete.*

PROOF. The PSPACE hardness follows from Proposition 7.3 as well as Proposition 5.1. We prove its membership in PSPACE in two steps: we first show that $\mathsf{SAT}(\mathcal{X}(\downarrow, \uparrow, \downarrow^*, \uparrow^*, \leftarrow, \rightarrow, \leftarrow^*, \rightarrow^*, \cup, [\ ], \neg))$ is in PSPACE under *nonrecursive* DTDs (Lemma 7.5 below). We then show that for fragments $\mathcal{X}(\cup, \ldots)$ containing union but *not* including the descendant and ancestor axes, there is a quadratic-time reduction from $\mathsf{SAT}(\mathcal{X}(\cup, \ldots))$ under *arbitrary* DTDs to $\mathsf{SAT}(\mathcal{X}(\cup, \ldots))$ under *nonrecursive* DTDs (Lemma 7.8). Putting these together, we get the PSPACE upper bound of $\mathsf{SAT}(\mathcal{X}(\downarrow, \uparrow, \leftarrow, \rightarrow, \leftarrow^*, \rightarrow^*, \cup, [\ ], \neg))$ under arbitrary DTDs. □

We first prove the PSPACE membership of $\mathsf{SAT}(\mathcal{X}(\downarrow, \uparrow, \downarrow^*, \uparrow^*, \leftarrow, \rightarrow, \leftarrow^*, \rightarrow^*, \cup, [\ ], \neg))$ under nonrecursive DTDs.

LEMMA 7.5. $\mathsf{SAT}(\mathcal{X}(\downarrow, \uparrow, \downarrow^*, \uparrow^*, \leftarrow, \rightarrow, \leftarrow^*, \rightarrow^*, \cup, [\ ], \neg))$ *is in PSPACE under nonrecursive DTDs.*

PROOF. The proof follows the approach used in Calvanese et al. [2001] for getting PSPACE bounds on queries with both backwards and forwards modalities transposed to the context of trees (Calvanese et al. [2001] deals with graph queries in the semi-structured data model). The idea is the following: we encode trees in terms of strings, using the standard tag representation. We will show that for a non-recursive DTD D, we can get (in polynomial time) a finite state machine $\mathcal{A}_D$ representing the set of codings of documents that conform to D (Claim 7.7). From a query p in $\mathcal{X}(\downarrow, \uparrow, \downarrow^*, \uparrow^*, \leftarrow, \rightarrow, \leftarrow^*, \rightarrow^*, \cup, [\ ], \neg)$, we will obtain in polynomial time an *alternating word automaton* $\mathcal{A}_p$ that accepts the strings coding a tree that satisfies p (Claim 7.6). We have thus reduced the satisfiability problem for this fragment to checking the non-emptiness of an alternating automaton (the conjunction of the alternating automata $\mathcal{A}_p$ and $\mathcal{A}_D$, representing the intersection of the languages accepted by these automata). The result will then follow from the fact that the emptiness problem for alternating word automata is known to be in PSPACE.

We first prove Claim 7.6 and then Claim 7.7.

7.3.1. *Tree Encoding.* We begin by discussing the coding of trees as words. First, we define the alphabet of a streamed document tree with element labels Σ as $\mathrm{XML}(\Sigma) = \{\langle A\rangle, \langle/A\rangle \mid A \in \Sigma\}$. For any tree T with element labels Σ, define $\mathrm{stream}(T) \in (\mathrm{XML}(\Sigma))^*$ as $\mathrm{stream}(root(T))$, where $root(T)$ is the root node of T and for arbitrary tree nodes n in T,

$$\mathrm{stream}(n) = \langle A\rangle \mathrm{stream}(n_1) \cdot \ \ldots \ \cdot \mathrm{stream}(n_k)\langle/A\rangle,$$

where $\mathsf{lab}(n) = A \in \Sigma$, $[n_1, n_2, \ldots, n_k]$ is the ordered list of children of n, and '$\cdot$' denotes string concatenation.

In order to identify a selected node in a tree, we will also consider streamed documents with an additional selected element. More specifically, we label the opening tag of the *selected node* with true, while all other opening tags are labeled with false. More formally, define the alphabet $\mathrm{XMLsel}(\Sigma) = \{(\langle A\rangle, \mathsf{false}), (\langle A\rangle, \mathsf{true}), \langle/A\rangle \mid A \in \Sigma\}$, and for a given (selected) node m in T define $\mathrm{stream}(T, m) \in (\mathrm{XMLsel}(\Sigma))^*$ as $\mathrm{stream}(root(T), m)$ where for arbitrary nodes n in T, $\mathrm{stream}(n, m)$ is

$$\begin{array}{ll} (\langle A\rangle, \mathsf{false})\ \mathrm{stream}(n_1, m) \cdots \mathrm{stream}(n_k, m)\langle/A\rangle & \text{if } \mathsf{lab}(n) = A \in \Sigma,\ n \neq m; \\ (\langle A\rangle, \mathsf{true})\ \mathrm{stream}(n_1, m) \cdots \mathrm{stream}(n_k, m)\langle/A\rangle & \text{if } \mathsf{lab}(n) = A \in \Sigma,\ n = m. \end{array}$$

As above, $[n_1, n_2, \ldots, n_k]$ is the ordered list of children of n.

7.3.2. *Two-Way Alternating Automata.* Our goal is to construct from an expression $p \in \mathcal{X}(\downarrow, \uparrow, \downarrow^*, \uparrow^*, \leftarrow, \rightarrow, \leftarrow^*, \rightarrow^*, \cup, [\ ], \neg)$ and a nonrecursive DTD D, an automaton that accepts exactly the set $\{\mathrm{stream}(T) \mid T \models (p, D)\}$. We now describe what sort of automaton we will need.

For a string s, we let $dom(s)$ be the set of positions in s, that is, $dom(s) = \{1, \ldots, |s|\}$, and for $i \in dom(s)$ we denote with $s(i)$ the label in s at position i.

Let $\mathrm{DIR} = \{\uparrow, \downarrow, \varepsilon\}$. For a string s, position $i \in dom(s)$, $\mathrm{dir} \in \mathrm{DIR}$ we let $i \cdot \mathrm{dir}$ be equal to the predecessor $i - 1$ of i if $\mathrm{dir} = \uparrow$, i if $\mathrm{dir} = \varepsilon$, and the successor $i + 1$ of i if $\mathrm{dir} = \downarrow$. The partial function $i \cdot \mathrm{dir}$ is undefined whenever the required predecessor or successor does not exist. For any set S let $B^+(S)$ be the set of

positive Boolean combinations over propositions P_i for $i \in S$, where we also allow the formulas true and false. A formula $\phi \in B^+(S)$ represents a collection of subsets of S in the usual manner. For instance, for a set $Q = \{q_i \mid i \in [1, n]\}$ of states, $q_1 \wedge q_2 \vee q_3$ is a formula in $B^+(Q)$.

A *two-way alternating automaton (2WAA)* is given by $(Q, \Sigma, \theta_0, \delta, F)$, where Q is the state space, Σ an alphabet, $\theta_0 \in B^+(Q)$, $F \subseteq Q$, and δ is a function from $Q \times \Sigma \rightarrow B^+(\text{DIR} \times Q)$.

Given a 2WAA $\mathcal{A}$, a string s over Σ, a *run* of $\mathcal{A}$ is a pair $\langle \mathcal{F}, \tau \rangle$ consisting of a finite forest $\mathcal{F} = (V, E)$ and a labeling function τ that assigns to each node of $\mathcal{F}$ a pair from $dom(s) \times Q$. Given $i_0 \in dom(s)$, a run *accepts* (s, i_0) if:

—The labels of the root nodes of $\mathcal{F}$ are $(i_0, q_0), \ldots, (i_0, q_n)$, where $\{q_0, \ldots, q_n\} \models \theta_0$;

—For any $x \in V$, letting $\tau(x) = (i, q)$ and $\delta(q, s(i)) = \theta$, there is a set $S \subseteq \text{DIR} \times Q$ such that $S \models \theta$ and for all $(\text{dir}, q') \in S$, there exists $y \in V$ such that $(x, y) \in E$ and $\tau(y) = (i \cdot \text{dir}, q')$; here S is referred to as a *satisfying assignment*; and

—For every leaf l of $\mathcal{F}$, $\tau(l) = (i, q)$ for some position $i \in dom(s)$ and state $q \in F$.

A 2WAA *accepts at* (s, i_0) if it has a run accepting (s, i_0).

We will also consider 2WAAs over the alphabet XMLsel(Σ), where Σ is the set of elements in the DTD. More specifically, for a set of element labels Σ, a *two-way alternating selection automaton* (2WASA) is simply an alternating automaton over XMLsel(Σ). We denote XMLsel(Σ) by Σ_{sel}. We will use an additional structure, a distinguished set of states $C \subseteq Q$, which we call the *critical states*. Intuitively, they represent states in the automaton where the automaton checks for the presence of a selected element. More formally, we consider 2WASAs of the form $\mathcal{A} = (Q, \Sigma_{sel}, \theta_0, \delta, F, C)$. Let N be an element label, $q \in Q$, and $\rho \subseteq \text{DIR} \times Q$. We say that ρ is a *select-demanding assignment* for (q, N) if $\rho \models \delta(q, (\langle N \rangle, \text{true}))$ but $\rho \not\models \delta(q, (\langle N \rangle, \text{false}))$. As we will show below, the set C of critical states will be the only states that have select-demanding assignments.

We remark that the notion of two-way alternating word automata that we use here is a mild extension of the standard one, allowing a *forest* instead of a *tree*. The use of forests is a convenience in the compositional translation of XPath. It is easy to move between this representation and the tree-based in polynomial time. Indeed, one can convert a forest-based 2WAA into a tree-based one by adding a new state that has self moves to all the initial states of the forest-based automaton.

7.3.3. *Translation of XPath Expressions into 2WA(S)As.* We first state the relationship between expressions in $\mathcal{X}(\downarrow, \uparrow, \downarrow^*, \uparrow^*, \leftarrow, \rightarrow, \leftarrow^*, \rightarrow^*, \cup, [\ ], \neg)$ and two-way alternating (selection) automata. More specifically, we define what it means for such automata (respectively, selection automata) to define unary (respectively, binary) relations on trees.

A 2WAA $\mathcal{A}$ over alphabet XML(Σ) is said to *define a unary relation* R on trees with element labels in Σ, if for every such tree T and node $n \in T$, $(T, n) \models R(n)$ iff $\mathcal{A}$ accepts at (stream(T), $pos(n)$), where $pos(n)$ denotes the position of the opening tag $\langle \text{lab}(n) \rangle$ of n in stream(T).

A 2WASA $\mathcal{A}$ over alphabet XMLsel(Σ) *defines a binary relation* R on trees with element labels in Σ, if for every such tree T and nodes $m, n \in T$, $(T, n, m) \models R(n, m)$ iff $\mathcal{A}$ accepts at (stream(T, m), $pos(n)$).

	$(\langle N\rangle, \mathsf{true})$	$(\langle N\rangle, \mathsf{false})$	$\langle/N\rangle$
q_0	$(\downarrow, q_1)$	$(\downarrow, q_1)$	false
q_1	(ϵ, q_f)	$(\downarrow, q_2)$	$(\downarrow, q_{i-1})$
$q_i (i \geq 2)$	$(\downarrow, q_{i+1})$	$(\downarrow, q_{i+1})$	$(\downarrow, q_{i-1})$
q_f	true	true	true

(a) Transition function for $\mathrm{trans}(\downarrow)$.

	$(\langle N\rangle, \mathsf{true})$	$(\langle N\rangle, \mathsf{false})$	$\langle/N\rangle$
q	$(\uparrow, q_0)$	$(\uparrow, q_0)$	false
q_0	(ϵ, q_f)	false	$(\uparrow, q_{i+1})$
$q_i (i \geq 1)$	$(\uparrow, q_{i-1})$	$(\uparrow, q_{i-1})$	$(\uparrow, q_{i+1})$
q_f	true	true	true

(b) Transition function for $\mathrm{trans}(\uparrow)$.

	$(\langle N\rangle, \mathsf{true})$	$(\langle N\rangle, \mathsf{false})$	$\langle/N\rangle$
q_0	$(\downarrow, q_1)$	$(\downarrow, q_1)$	false
q_1	$(\downarrow, q_2)$	$(\downarrow, q_2)$	$(\downarrow, q)$
q	(ϵ, q_f)	false	false
$q_i (i \geq 2)$	$(\downarrow, q_{i+1})$	$(\downarrow, q_{i+1})$	$(\downarrow, q_{i-1})$
q_f	true	true	true

(c) Transition function for $\mathrm{trans}(\rightarrow)$.

	$(\langle N\rangle, \mathsf{true})$	$(\langle N\rangle, \mathsf{false})$	$\langle/N\rangle$
q_0	(ϵ, q_f)	$(\downarrow, q_1)$	false
$q_i (i \geq 1)$	(ϵ, q_f)	$(\downarrow, q_{i+1})$	$(\downarrow, q_{i-1})$
q_f	true	true	true

(d) Transition function for $\mathrm{trans}(\downarrow^*)$.

FIG. 10. Transitions functions for the 2WASA corresponding to some of the XPath axes.

In a nutshell, we will use 2WAAs to characterize XPath qualifiers and 2WASAs to encode node-selecting XPath queries. Recall that an XPath query p determines a binary relation R. To check whether or not $(T, n, m) \models R(n, m)$ for nodes n, m in a tree with element labels in Σ, we convert T to a tree with element labels in XMLsel(Σ), also denoted by T, by labeling the opening tag of m with true and all other opening tags with false. We then use the 2WASA $\mathcal{A}$ characterizing p to determine whether or not $\mathcal{A}$ accepts at (stream(T, m), $pos(n)$).

More formally, this is stated in the main claim of the proof.

CLAIM 7.6. *Given sets of elements Σ, there are linear-time functions* trans *and* qtrans *such that:*

—trans *takes an $\mathcal{X}(\downarrow, \uparrow, \downarrow^*, \uparrow^*, \leftarrow, \rightarrow, \leftarrow^*, \rightarrow^*, \cup, [\,], \neg)$ expression p mentioning only elements in Σ, and returns a two-way alternating selection automaton that defines the same binary relation on trees over Σ as p does;*

—qtrans *takes each $\mathcal{X}(\downarrow, \uparrow, \downarrow^*, \uparrow^*, \leftarrow, \rightarrow, \leftarrow^*, \rightarrow^*, \cup, [\ ], \neg)$ qualifier q mentioning only elements in Σ, and returns a two-way alternating automaton that defines the same unary relation on trees over Σ as q does.*

PROOF OF CLAIM. We prove Claim 7.6 by defining the functions trans and qtrans by induction on the structure of p and q, followed by the correctness proof of the translation.

Recall that we only consider non-recursive DTDs. Hence, provided that we are dealing with a string that codes a tree satisfying a non-recursive DTD, we know that if a start tag $\langle N\rangle$, $(\langle N\rangle, \mathsf{true})$, or $(\langle N\rangle, \mathsf{false})$ appears, then sometime afterwards, the corresponding end tag $\langle/N\rangle$ must appear. Moreover, in between these two tags there are *no* additional occurrences of $\langle N\rangle$ or $\langle/N\rangle$. Another consequence of using nonrecursive DTDs is that we can calculate the maximal depth n of any element label in a tree satisfying the DTD. We will use this bound in the construction below.

We now define functions trans and qtrans.

Induction basis. First, we consider base cases for queries and qualifiers, beginning with the navigational axes.

(1) $p = \downarrow$: Then $\mathrm{trans}(\downarrow) = (Q, \Sigma_{sel}, \theta_0, \delta, F, C)$, where $Q = \{q_0, q_1, \ldots, q_n, q_f\}$, $\theta_0 = q_0$, and δ as defined in Figure 10(a). Moreover, $F = \{q_f\}$ and $C = \{q_1\}$. As depicted in Figure 11, only state q_1 leads to an

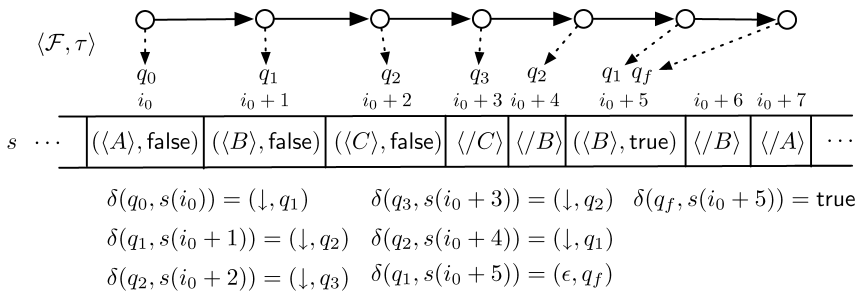

FIG. 11. Illustration of a run $\langle \mathcal{F}, \tau \rangle$ of trans($\downarrow$) on a string s. For convenience, we also include the transition functions used in this run.

accepting state (q_f), whereas for $2 \leq i \leq n$, states q_i's are used to skip descendants of the context node v. More specifically, when the start tag ($\langle N \rangle$, *val*) of a descendant u of v is encountered at state q_i, $\delta(q_i, (\langle N \rangle, val))$ moves to the next state q_{i+1}. When the end tag $\langle /N \rangle$ of u is encountered at state q_i, $\delta(q_i, \langle /N \rangle)$ moves to the previous state q_{i-1}, indicating that the element u has been passed and the 2WASA has moved to a symbol corresponding to the next level up in the tree. Note that trans($\downarrow$) needs states $q_1, \ldots, q_n$ since the maximal depth of the DTD is n. Here q_1 is the only critical state.

(2) $p = \uparrow$: Then trans($\uparrow$) $= (Q, \Sigma_{sel}, \theta_0, \delta, F, C)$, where $Q = \{q, q_0, q_1, \ldots, q_n, q_f\}$, $\theta_0 = q$, and δ is as defined in Figure 10(b). The intuition behind δ is similar to case (1) above. Moreover, $F = \{q_f\}$ and $C = \{q_0\}$.

(3) $p = \rightarrow$: Then trans($\rightarrow$) $= (Q, \Sigma_{sel}, \theta_0, \delta, F, C)$, where $Q = \{q, q_0, q_1, \ldots, q_n, q_f\}$, $\theta_0 = q_0$, and δ is as defined in Figure 10(c). The intuition behind δ is similar to case (1) above. Moreover, $F = \{q_f\}$ and $C = \{q\}$.

(4) $p = \downarrow^*$: Then trans($\downarrow^*$) $= (Q, \Sigma_{sel}, \theta_0, \delta, F)$, where $Q = \{q_0, q_1, \ldots, q_n, q_f\}$, $\theta_0 = q_0$ and δ is as defined in Figure 10(d). The intuition behind δ is similar to case (1) above. Moreover, $F = \{q_f\}$ and $C = \{q_0, q_1, \ldots, q_n\}$. Note that in this case the set C consists of multiple critical states.

(5) The other base cases, ε, A, $\leftarrow$, $\uparrow^*$, $\leftarrow^*$ and $\rightarrow^*$ are dealt with in a similar way.

Next, we consider qualifiers $[q]$.

(6) $q =$ "lab() $= A$": Then qtrans(q) $= (Q, \Sigma, \theta_0, \delta, F)$, where $Q = \{q_0, q_f\}$, $\theta_0 = q_0$ and $\delta(q_0, \langle A \rangle) = (\varepsilon, q_f)$, $\delta(q_f, \alpha) =$ true for every $\alpha \in \Sigma$ and $\delta(q, \alpha) =$ false for all other $(q, \alpha) \in Q \times \Sigma$. Finally, $F = \{q_f\}$.

(7) $q = q_1 \wedge q_2$: Suppose that qtrans(q_1) $= (Q, \Sigma, \theta_0, \delta, F)$ and qtrans(q_2) $= (Q', \Sigma, \theta_0', \delta', F')$ with $Q \cap Q' = \emptyset$. Then qtrans($q_1 \wedge q_2$) $= (Q'', \Sigma, \theta_0'', \delta'', F'')$, where $Q'' = Q \cup Q'$, $\theta_0'' = \theta_0 \wedge \theta_0'$, $\delta'' = \delta \cup \delta'$, and $F'' = F \cup F'$.

(8) $q = \neg q_1$: Suppose that qtrans(q_1) $= (Q, \Sigma, \theta_0, \delta, F)$ then qtrans($\neg q_1$) $= (Q', \Sigma, \theta_0', \delta', F')$, where $Q' = Q$, $\theta_0' =$ Dual(θ_0), $\delta'(q, \alpha) =$ Dual($\delta(q, \alpha)$) for $\alpha \in \Sigma$, $F' = Q' - F$. Here, for $\theta \in B^+(\text{DIR} \times Q)$, Dual($\theta$) is defined by induction on the structure of θ: Dual(false) $=$ true, Dual(true) $=$ false, Dual($\bigvee \theta_i$) $= \bigwedge \theta_i$, Dual($\bigwedge \theta_i$) $= \bigvee \theta_i$.

(9) $q = p$: Suppose that trans(p) $= (Q, \Sigma_{sel}, \theta_0, \delta, F, C)$. Then qtrans($p$), where p is considered as a qualifier, is given by $(Q', \Sigma, \theta_0', \delta', F')$, where $Q' = Q$, $\theta_0' = \theta_0$, and for all $q \in Q'$ and labels N, $\delta'(q, \langle N \rangle) = \delta(q, (\langle N \rangle, \text{false})) \vee \delta(q, (\langle N \rangle, \text{true}))$ and $\delta'(q, \langle /N \rangle) = \delta(q, \langle /N \rangle)$. This is because for qualifiers the 2WASA ignores

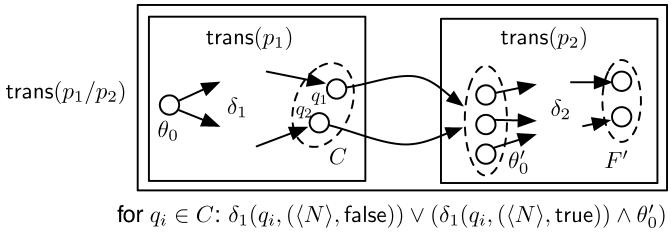

FIG. 12. Illustration of the construction of trans(p_1/p_2). Critical states in trans(p_1) are conjoined with the initial states of trans(p_2).

the labels true and false, since these labels are only used for selecting nodes. Finally, $F' = F$.

Induction Steps for Expressions. We next consider the induction cases for expressions. Suppose that trans(p_1) $= (Q, \Sigma_{sel}, \theta_0, \delta, F, C)$ and trans(p_2) $= (Q', \Sigma_{sel}, \theta'_0, \delta', F', C')$ such that $Q \cap Q' = \emptyset$.

(1) $p = p_1 \cup p_2$: Then trans($p_1 \cup p_2$) $= (Q'', \Sigma_{sel}, \theta''_0, \delta'', F'', C'')$, where $Q'' = Q \cup Q'$, $\theta''_0 = \theta_0 \vee \theta'_0$, $\delta'' = \delta \cup \delta'$, $F'' = F \cup F'$ and $C'' = C \cup C'$.

(2) $p = p_1/p_2$: Then trans(p_1/p_2) $= (Q'', \Sigma_{sel}, \theta''_0, \delta'', F'', C'')$, where $Q'' = Q \cup Q'$, $\theta''_0 = \theta_0$. For the definition of the transition function δ, we consider the following cases:

—$q \in C$, $(\langle N\rangle, \mathsf{false}) \in \Sigma_{sel}$: then $\delta''(q, (\langle N\rangle, \mathsf{false})) = \delta(q, (\langle N\rangle, \mathsf{false})) \vee (\delta(q, (\langle N\rangle, \mathsf{true})) \wedge \theta_0^{\varepsilon}))$, θ_0^{ε} is obtained from θ'_0 by changing every occurrence of a state q' in θ'_0 to (ε, q');

—$q \in Q - C$, $(\langle N\rangle, \mathsf{false}) \in \Sigma_{sel}$: then $\delta''(q, (\langle N\rangle, \mathsf{false})) = \delta(q, (\langle N\rangle, \mathsf{false}))$;

—$q \in Q$, $(\langle N\rangle, \mathsf{true}) \in \Sigma_{sel}$: then $\delta''(q, (\langle N\rangle, \mathsf{true})) = \delta''(q, (\langle N\rangle, \mathsf{false}))$;

—$q \in Q$, $\langle /N\rangle \in \Sigma_{sel}$: then $\delta''(q, \langle /N\rangle) = \delta(q, \langle /N\rangle)$; and

—$q' \in Q'$, $\alpha \in \Sigma_{sel}$: then $\delta''(q', \alpha) = \delta(q', \alpha)$.

Moreover, $F'' = F'$ and $C'' = C'$.

As shown in Figure 12, trans(p_1/p_2) is constructed by conjoining the *critical states* of trans(p_1) and the start states of trans(p_2). This is why we keep track of critical states of 2WASAs. Since only the selected nodes are marked true and trans(p_1/p_2) selects nodes different from those of trans(p_1), we define $\delta''(q, (\langle N\rangle, \mathsf{false}))$ to be $\delta(q, (\langle N\rangle, \mathsf{false})) \vee (\delta(q, (\langle N\rangle, \mathsf{true})) \wedge \theta_0^{\varepsilon}))$ for the critical states of trans(p_1), such that trans(p_1/p_2) now treats the true and false labels of those critical states of trans(p_1) uniformly and indifferently. For the same reason, we define $\delta''(q, (\langle N\rangle, \mathsf{true})) = \delta''(q, (\langle N\rangle, \mathsf{false}))$ for any state q in trans(p_1). Note that trans(p_1/p_2) inherits the final states and critical states of trans(p_2).

(3) $p = p_1[q]$: Suppose that qtrans(q) $= (Q', \theta'_0, \delta', F')$. Then, we have trans($p[q]$) $= (Q'', \Sigma_{sel}, \theta''_0, \delta'', F'', C'')$, where $Q'' = Q \cup Q'$, $\theta''_0 = \theta_0$. For the definition of the transition function δ, we distinguish the following cases; the intuition behind the definition is similar to the case of p_1/p_2 described above.

—$q \in C$, $(\langle N\rangle, \mathsf{true}) \in \Sigma_{sel}$: then $\delta''(q, (\langle N\rangle, \mathsf{true})) = \delta(q, (\langle N\rangle, \mathsf{true})) \wedge \theta_0^{\varepsilon}$, where θ_0^{ε} is obtained from θ'_0 by changing every occurrence of a state q' in θ'_0 to (ε, q');

—$q \in Q - C$, $(\langle N\rangle, \mathsf{true}) \in \Sigma_{sel}$: then we have $\delta''(q, (\langle N\rangle, \mathsf{true})) = \delta(q, (\langle N\rangle, \mathsf{true}))$;

—$q \in Q$, $(\langle N\rangle, \mathsf{false}) \in \Sigma_{sel}$: then $\delta''(q, (\langle N\rangle, \mathsf{false})) = \delta(q, (\langle N\rangle, \mathsf{false}))$;
—$q \in Q$, $\langle /N\rangle \in \Sigma_{sel}$: then $\delta''(q, \langle /N\rangle) = \delta(q, \langle /N\rangle)$;
—$q' \in Q'$ and $(\langle N\rangle, \mathsf{false}), (\langle N\rangle, \mathsf{true}) \in \Sigma_{sel}$: then $\delta''(q', (\langle N\rangle, \mathsf{false})) = \delta''(q', (\langle N\rangle, \mathsf{true})) = \delta(q', \langle N\rangle)$; and
—$q' \in Q'$, $\langle /N\rangle \in \Sigma_{sel}$: then $\delta''(q', \langle /N\rangle) = \delta'(q', \langle /N\rangle)$.

Moreover, $F'' = F'$ and $C'' = C$.

7.3.4. *Properties of Accepting (Pseudo-)Runs.* To prove the correctness of the 2WASA construction, we will need some properties of 2WASA runs; these properties will assure that the "projection steps" (e.g. for qualifiers of the form $[p]$, where p is an expression) are sound. Projection requires us to know more about accepting runs of the 2WASAs we generate, and more generally for accepting *pseudo-runs* of 2WASAs. Intuitively, pseudo-runs ignore the selected elements in a stream of an XML tree.

Given a 2WASA $\mathcal{A} = (Q, \Sigma_{sel}, \theta_0, \delta, F, C)$ and a string s, a *pseudo-run* of $\mathcal{A}$ is a pair $\langle \mathcal{F}, \tau\rangle$ consisting of a finite labeled forest $\mathcal{F} = (V, E)$ and a labeling function τ that assigns to each node of $\mathcal{F}$ a pair from $dom(s) \times Q$. Given $i_0 \in dom(s)$, a pseudo-run *accepts* (s, i_0) if:

—The labels of the root nodes of $\mathcal{F}$ are $(i_0, q_0), \ldots, (i_0, q_n)$, where $\{q_0, \ldots, q_n\} \models \theta_0$;
—For any $x \in V$ such that $\tau(x) = (i, q)$, either (a) $s(i) = \langle /N\rangle$ for some N, and letting $\delta(q, s(i)) = \theta$, there is a set $S \subseteq \text{DIR} \times Q$ such that $S \models \theta$, and for all $(\text{dir}, q') \in S$ there exists a node $y \in V$ such that $(x, y) \in E$ and $\tau(y) = (i \cdot \text{dir}, q')$; or (b) $s(i) = (\langle N\rangle, \mathsf{true})$ or $s(i) = (\langle N\rangle, \mathsf{false})$ and for some $val \in \{\mathsf{true}, \mathsf{false}\}$, letting $\delta(q, (\langle N\rangle, val)) = \theta$, there is a set $S \subseteq \text{DIR} \times Q$ such that $S \models \theta$, and for all $(\text{dir}, q') \in S$ there exists a node $y \in V$ such that $(x, y) \in E$ and $\tau(y) = (i \cdot \text{dir}, q')$; and
—For every leaf l of $\mathcal{F}$, $\tau(l) = (i, q)$ for some position $i \in dom(s)$ and state $q \in F$.

In contrast to a run, a pseudo-run treats true and false indifferently.

We say that $\langle \mathcal{F}, \tau\rangle$ is a *minimal* pseudo-run of $\mathcal{A}$ accepting (s, i_0) if it is a pseudo-run accepting (s, i_0) and moreover, there is no pseudo-run $\langle \mathcal{F}', \tau'\rangle$ of $\mathcal{A}$ such that $\mathcal{F}'$ is a sub-structure of $\mathcal{F}$.

We now present the properties of a 2WASA $\mathcal{A} = \text{trans}(p)$.

PROPERTY 1 (**(P1)**). *Let T be a tree that satisfies a nonrecursive DTD D, m a node in T, $\langle \mathcal{F}, \tau\rangle$ a minimal pseudo-run of $\mathcal{A}$ that accepts* $(\text{stream}(T, m), i_0)$. *Then, there is some position i in the string* $\text{stream}(T, m)$, *some vertex v in $\mathcal{F}$ such that $\tau(v) = (i, q)$, $q \in C$, $s(i) = (\langle N\rangle, val)$ for some element label N and $val \in \{\mathsf{true}, \mathsf{false}\}$, and the children of v in $\mathcal{F}$ satisfy $\delta(q, (\langle N\rangle, \mathsf{true}))$.*

That is, every pseudo-run of the automaton reaches a critical state on a node that could be selected. We show that (P1) holds by induction on the structure of p.

To see that (P1) holds for all the base cases, note that the automata for all the axes have nontrivial initial conditions and as a result, an accepting pseudo-run cannot be empty. In a nonempty pseudo-run, the leaf nodes must be labeled with accepting states, and in the automata for the axes the accepting state is only reachable through a transition from a critical state q through $\delta(q, (\langle N\rangle, \mathsf{true}))$ for some element label N. From this, it follows that (P1) holds for all the base cases.

We now show that (P1) is preserved inductively. For union, this is clear, since a pseudo-run for the union is a pseudo-run for one of its components. Now consider the case of $p = p_1/p_2$. We can see that there are no nonempty pseudo-runs. Indeed, by induction the initial conditions are never vacuously satisfied for all of the automata generated by expressions. By construction, since the accepting states of the automaton trans(p_1/p_2) are the accepting states of trans(p_2), a pseudo-run for trans(p_1/p_2) must contain a pseudo-run for trans(p_2) as a subtree. Hence, (P1) holds by the induction hypothesis. Finally, considering the case of $p = p_1[q]$, we see (by construction) that a pseudo-run must contain a pseudo-run for the automaton trans(p_1) as a subtree, and the result follows by induction.

PROPERTY 2 ((**P2**)). *Let $\langle \mathcal{F}, \tau \rangle$ be a minimal pseudo-run as described in Property* I. *If v_1 and v_2 are two vertices of the forest $\mathcal{F}$, v_2 is a descendant of v_1 in $\mathcal{F}$, $\tau(v_1) = (i_1, q_1)$, $\tau(v_2) = (i_2, q_2)$ and both q_1 and q_2 are critical states, then $i_1 \neq i_2$.*

That is, in every pseudo-run of the automaton, we cannot get to a critical state twice on the same position in the stream.

Again, we show that (P2) holds by induction on the structure of p.

To see that (P2) holds for all the base cases, note that in the automata for all the axes, the outgoing transitions of a critical state are either into a state that does not reach a critical state, or are transitions into another critical state but in a direction that is not ε and depends only on the axis, not the state. For instance, outgoing transitions of a critical state for the child axis all move downward, for the parent axis they all move upwards, and so on. Thus (P2) holds for the base cases.

For the inductive cases, preservation under union is clear, since a pseudo-run for the union is a pseudo-run for one of the components. For $p = p_1/p_2$, we observe that the critical states are those of p_2, with their transitions the same as in trans(p_2). Hence, (P2) is preserved by induction. In the case of $p = p_1[q]$, it is clear from the construction that a pseudo-run for the automaton trans($p_1[q]$) contains a pseudo-run for trans(p_1) as a subtree, with the critical states lying entirely within this sub-pseudo-run. Hence, (P2) is preserved by induction.

Let $A = (Q, \Sigma_{sel}, \theta_0, \delta, F, C)$ be a 2WASA, N an element label, $q \in Q$, and $\rho \subseteq \text{DIR} \times Q$. Recall that we say ρ is a *select-demanding assignment* for (q, N) if $\rho \models \delta(q, (\langle N \rangle, \mathsf{true}))$ but $\rho \not\models \delta(q, (\langle N \rangle, \mathsf{false}))$. We say that ρ is a *select-denying assignment* for (q, N) if $\rho \models \delta(q, (\langle N \rangle, \mathsf{false}))$ and $\rho \not\models \delta(q, (\langle N \rangle, \mathsf{true}))$. We say that a state $q \in Q$ is *select-indifferent* if there are no select-denying or select-demanding assignments for (q, N) for any element label N, and *transitively select-indifferent* if all states reachable from q via the transition relation are select-indifferent.

Note that each pseudo-run assigns a set $S \subseteq \text{DIR} \times Q$ to each pair (q, N), referred to as assignments induced by the run.

PROPERTY 3 ((**P3**)). *Let $\langle \mathcal{F}, \tau \rangle$ be a minimal pseudo-run as in Property 1. If this run induces a select-demanding or select-denying assignment S for some pair (q, N), then q is a critical state.*

That is, all states that either require or forbid selection are critical states.

We show that (P3) holds by induction on the structure of p. All of these properties are easily seen to hold in the base cases for expressions (i.e., all the axes). They

are also easily seen to be preserved inductively in the case of the union operator. For the case that $p = p_1/p_2$ observe that for select demanding states of trans(p_2), they are critical states by induction. For states of trans(p_1), note that none of them can have a select-demanding transition, since the transition function is symmetric in true and false by the definition of trans(p_1/p_2). Hence, the requirement is vacuously true for assignments to these states. Similarly, consider the case $p = p_1[q]$. For this, we observe that only states for which the transitions are not symmetric in true and false are the critical states of trans(p_1), and thus the result follows.

PROPERTY 4 ((**P4**)). *Let $\langle \mathcal{F}, \tau \rangle$ be a minimal pseudo-run as in Property 1. If this run induces a satisfying assignment ρ for some formula $\delta(q, (\langle N \rangle, \mathsf{true}))$ and $q \in C$, then all the states mentioned in ρ do not reach a critical state via the transition relation. In particular, by* (P3)*, all the states reachable via such a transition are select-indifferent.*

That is, once a run hits a critical state and selects a node, it cannot be forced to select it again.

We show that (P4) holds by induction on the structure of p. We omit the base cases and union for which (P4) is easily verified. Consider $p = p_1/p_2$. Notice that the critical states of trans(p_1/p_2) are those of trans(p_2), and the transitions out of these states are the same as those of trans(p_2). Hence, (P4) holds by induction. Next, consider $p = p_1[q]$. The critical states are those for trans(p_1). The transitions out of these go to a conjunction of either states in trans(p_1) or states in trans(q). The states in trans(q) are select-indifferent by the construction (they do not distinguish true and false). Moreover, since states in trans(q) move into states in trans(q), they are transitively select-indifferent. For each state in trans(p), if it is not a critical state, then its transition function within trans(p_1/p_2) is the same as within trans(p_1). Hence, (P4) holds by the induction hypothesis.

PROPERTY 5 ((**P5**)). *Let $\langle \mathcal{F}, \tau \rangle$ be a minimal pseudo-run as in Property* 1*. If this run induces a satisfying assignment ρ for some formula $\delta(q, \alpha)$, then at most one* $(\mathrm{dir}, q) \in \rho$ *reaches a critical state (and hence the other reachable states are transitively select-indifferent). Similarly, if the run induces a satisfying assignment ρ for the initial formula θ_0 of the automaton, then at most one $q \in \rho$ reaches any critical state.*

That is, there cannot be parallel threads that all demand the presence of a selecting element.

We show that (P5) holds by induction on the structure of p. We observe that if a transition does not contain a conjunction, then all minimal satisfying subsets will be singletons, and hence (P5) is vacuously true. This implies that (P5) holds in the base cases (which do not use conjunction) and in the inductive case for union, since conjunction is not introduced there.

Consider the case of $p = p_1/p_2$. In this case, conjunction is introduced only in the transitions for states in trans(p_1). Hence, for states in trans(p_2), the result follows by induction. For the states in trans(p_1), the introduced conjunctions are of the form $\delta(q, (\langle N \rangle, \mathsf{true})) \wedge \theta_0^{\varepsilon}$. Moreover, these are transitions both for $(\langle N \rangle, \mathsf{true})$ and $(\langle N \rangle, \mathsf{false})$. In a minimal satisfying assignment ρ for such a transition, we can partition the pairs into two sets ρ_0 and ρ_1, with $\rho_0 \models \delta(q, (\langle N \rangle, \mathsf{true}))$ and $\rho_1 \models \theta_0^{\varepsilon}$. States reachable from ρ_1 do not distinguish true from false, hence all are transitively select-indifferent. Since the conjunction above is the same for true as

for false, one can see that a state in trans(p_1) is select-indifferent in trans(p_1/p_2) iff it is select-indifferent in the automaton trans(p_1). Hence, (P5) holds by induction.

Next, consider $p = p_1[q]$. All states of trans(q) are transitively select-indifferent, so one needs to consider only states of trans(p_1). The noncritical states of trans(p_1) have exactly the same transition as they do in trans(p); hence, the property (P5) holds by induction for these states. We next consider the critical states of trans(p_1). Let ρ be a minimal satisfying assignment for a transition out of a critical state of trans($p_1[q]$). As in the previous case, we partition ρ into ρ_0 and ρ_1. By the definition of trans(p), states reachable from ρ_1 do not distinguish true from false, hence all are transitively select-indifferent. Moreover, the states reachable from ρ_0 are select-indifferent by invariant (P4). Thus, the statement holds in this case.

7.3.5. *Correctness of the Translation.* We are now ready to prove Claim 7.6. The proof is by induction on the structure of p.

Induction basis. It is easy to verify the claim for the base cases. We next concentrate on the inductive cases.

Induction step. The inductive cases for boolean operators in qualifiers and for the union operator are straightforward.

$p = p_1/p_2$: Suppose that we have T, m, n such that $T \models p(n, m)$. Then, there exists $n' \in T$ such that $T \models p_1(n, n')$ and $T \models p_2(n', m)$. By induction, we have an accepting run $\langle \mathcal{F}_1, \tau_1 \rangle$ for trans(p_1) on the string stream(T, n') at position $pos(n)$. Similarly, we have an accepting run $\langle \mathcal{F}_2, \tau_2 \rangle$ for trans(p_2) on the string stream(T, m) at position $pos(n')$.

Since $\mathcal{F}_1$ is an accepting run, and hence an accepting pseudo-run, by (P1) there must be a vertex v such that $\tau_1(v) = (q, pos(n'))$ with q a critical state of trans(p_1), and moreover, the children of v satisfy $\delta(q, (\langle N \rangle, \mathsf{true}))$, where $(\langle N \rangle, \mathsf{true})$ is the letter at position $pos(n')$ in stream(T, n') (i.e., the selected element).

We now construct a new labeled forest $\langle \mathcal{F}, \tau \rangle$ by appending $\mathcal{F}_2$ to $\mathcal{F}_1$ at vertex v and letting $\tau = \tau_1 \cup \tau_2$. We claim that this is an accepting run for trans(p_1/p_2) on stream(T, m) at position $pos(n)$. This claim follows from the construction. Indeed, $\langle \mathcal{F}, \tau \rangle$ is an accepting run because each node of trans(p_1) becomes select-indifferent within trans(p_1/p_2). Hence, the transitions of $\langle \mathcal{F}_1, \tau_1 \rangle$ are valid for $\langle \mathcal{F}, \tau \rangle$ even though the selected node, that is, m, is different from that of the input for $\langle \mathcal{F}_1, \tau_1 \rangle$, that is, n'. Furthermore, $\langle \mathcal{F}, \tau \rangle$ behaves the same as $\langle \mathcal{F}_2, \tau_2 \rangle$ starting from v.

In the other direction, suppose that we have T and nodes m, n and assume that trans(p_1/p_2) accepts on stream(T, m) starting at $pos(n)$. Let $\langle \mathcal{F}, \tau \rangle$ be a minimal accepting run that witnesses this.

It is clear from the initial condition of the automaton that $\langle \mathcal{F}, \tau \rangle$ must be nonempty, and that all of its leaf nodes must be accepting states of trans(p_2). From (P1) we know that $\langle \mathcal{F}, \tau \rangle$ must reach a critical state, and all such critical states are states of trans(p_2). Hence, at some point there must be a transition of states in trans(p_1) into states in trans(p_2). Let v be a node in $\mathcal{F}$ of minimal depth such that v has children that are labeled with states of trans(p_2). Let $n' \in T$ be the node corresponding to the ith position in stream(T, m), where $\tau(v) = (q, i)$. From the definition of trans(p_1/p_2), it is clear that v is a critical state of trans(p_1) and the subtree in $\mathcal{F}$ below v represents an accepting run of trans(p_2). Hence, by induction, $T \models p_2(n', m)$.

Consider next the run $\langle \mathcal{F}_1, \tau_1 \rangle$ formed from $\langle \mathcal{F}, \tau \rangle$ as follows: remove all vertices of $\mathcal{F}$ associated by τ with states of trans(p_2), and let τ_1 be the restriction of τ to this sub-forest. $\langle \mathcal{F}_1, \tau_1 \rangle$ is a run on stream(T, n') starting at $pos(n)$, and we show that it is an accepting run, which will imply that $T \models p_1(n, n')$. It is clear from the construction that $\langle \mathcal{F}_1, \tau_1 \rangle$ is a pseudo-run of trans(p_1) accepting stream(T, n') at $pos(n)$, since every transition must correspond to a transition of trans(p_1) for either true or false. What remains to argue is that in changing the selected node from n to n', we did not destroy the acceptance of $\langle \mathcal{F}, \tau \rangle$. Consider nodes in $\mathcal{F}_1$. The construction of the automaton guarantees that the children of v in $\langle \mathcal{F}_1, \tau_1 \rangle$ satisfy the transition of trans(p_1). From (P4), it follows that the descendants of v in $\langle \mathcal{F}_1, \tau_1 \rangle$ must be transitively select-indifferent states of trans(p_1). Hence, the subtree below v will still satisfy the transition function for trans(p_1) on stream(T, n'). Now consider the nodes above v in $\langle \mathcal{F}_1, \tau_1 \rangle$. If they are not critical states of trans(p_1), then they are select-indifferent and hence will satisfy the appropriate transition. If they are critical states of trans(p_1), then by (P2) they cannot be labeled with the same position as v. Since they have a descendent that corresponds to a critical state (namely v), it follows from (P4) that their children cannot satisfy the disjunct of the transition in trans(p_1/p_2) that corresponds to the selection (recall the definition of trans(p_1/p_2)); this is because such a transition would lead only to transitively select-indifferent states. Hence, since $\langle \mathcal{F}, \tau \rangle$ was an accepting run, the children of such a node must have satisfied the disjunct corresponding to nonselection. Since this node is not the selected node, this means that the children satisfy the transition formula for trans(p_1) on stream(T, n'). Finally, for the nodes that are neither ancestors nor descendants of v, they cannot be in a critical state of trans(p_1), by property (P5), and hence they are select-indifferent for trans(p_1). Hence, the run $\langle \mathcal{F}_1, \tau_1 \rangle$ must be an accepting run of trans(p_1) on stream(T, n'), as required.

$p = p_1[q]$: Suppose that we have T, m, n such that $T \models p_1[q](n, m)$. We first show that trans($p_1[q]$) accepts stream(T, m) at position $pos(n)$. Note that $T \models p_1(n, m)$ and by induction trans(p_1) accepts stream(T, m) at position $pos(n)$. Let $\langle \mathcal{F}_1, \tau_1 \rangle$ be an accepting run of trans(p_1) that witnesses this. From property (P1), we know that there exists a vertex $v \in \mathcal{F}_1$ labeled with $(q, pos(m))$, where q is a critical state such that the children of v in $\mathcal{F}_1$ satisfy $\delta(q, (\langle N \rangle, \mathsf{true}))$, in which $(\langle N \rangle, \mathsf{true})$ is the letter at position $pos(m)$ in stream(T, m).

Similarly, let $\langle \mathcal{F}_2, \tau_2 \rangle$ be a run of qtrans(q) which accepts stream(T) at position $pos(m)$. Let $\langle \mathcal{F}, \tau \rangle$ be the run obtained by attaching $\mathcal{F}_2$ to $\mathcal{F}_1$ at v and letting $\tau = \tau_1 \cup \tau_2$. Then, $\langle \mathcal{F}, \tau \rangle$ is an accepting run for the automaton trans($p_1[q]$). Indeed, this follows from the construction of trans($p_1[q]$) and the fact that the nodes of qtrans(q) are transitively select-indifferent by construction.

Conversely, suppose that we have T, m, n as above and that trans($p_1[q]$) accepts stream(T, m) at position $pos(n)$. Let $\langle \mathcal{F}, \tau \rangle$ be a run that witnesses this. We show that $T \models p_1[q](n, m)$. It is clear from the construction that $\langle \mathcal{F}, \tau \rangle$ can be pruned to become an accepting run for trans(p_1) by eliminating the subtree of nodes labeled with states in trans(q). Hence, by induction, $T \models p_1(n, m)$. Moreover, from property (P1), we know that $\langle \mathcal{F}, \tau \rangle$ must reach a critical state with a node that could be selected. Looking at the transition for critical states, we see that there must be a subtree of the run which is an accepting run of trans(q) starting at $pos(m)$. Putting these together, we have that $T \models p_1[q](n, m)$.

We next consider the case of qualifiers $q = p$.

Consider the 2WAA qtrans(p), where p is regarded as a qualifier. Suppose that qtrans(p) accepts stream(T) at position $pos(n)$ for some node $n \in T$. Let $\langle \mathcal{F}, \tau \rangle$ be a minimal accepting run of qtrans(p) that witnesses this. We show that $T \models q(n)$.

Clearly, $\langle \mathcal{F}, \tau \rangle$ can be regarded as a pseudo-run for trans(p), when p is regarded as a normal expression, that is, not a qualifier. By property (P1), there is some vertex v in $\mathcal{F}$ such that $\tau(v) = (pos(m), q')$, where q' is a critical state such that the children of v in $\mathcal{F}$ satisfy $\delta(q', (\langle N \rangle, \mathsf{true}))$, in which $(\langle N \rangle, \mathsf{true})$ is the letter at position $pos(m)$ in stream(T, m) for some node $m \in T$. Choose the highest such node v in $\mathcal{F}$.

To show that $T \models p(n, m)$, it is sufficient (by induction) to show that $\langle \mathcal{F}, \tau \rangle$ is an accepting run for trans(p), where p is viewed as a normal expression, on stream(T, m) at position $pos(n)$.

Consider a vertex w of $\mathcal{F}$ such that $\tau(w) = (q'', i)$. Recall that N is the element label of node m in T. Since N is the only tag labeled with true in stream(T, m) it is sufficient to look at transitions involving this tag. By the definition of the automaton qtrans(p) and the fact that $\langle \mathcal{F}, \tau \rangle$ is an accepting run, we know that the successors of w satisfy $\delta'(q'', \langle N \rangle) = \delta(q'', (\langle N \rangle, \mathsf{false})) \vee \delta(q'', (\langle N \rangle, \mathsf{true}))$. Here δ' and δ denote the transition function of qtrans(p) and trans(p), respectively.

To show that $\langle \mathcal{F}, \tau \rangle$ is an accepting run for trans(p) on stream(T, m) at position $pos(n)$, we need to show that successors of w satisfy $\delta(q'', (\langle N \rangle, \mathsf{false}))$ if $i \neq pos(m)$, and satisfy $\delta(q'', (\langle N \rangle, \mathsf{true}))$ if $i = pos(m)$.

If $w = v$, then we know that this is true by the choice of v. Next, we consider the case where w is an ancestor of v in $\mathcal{F}$. If q'' is a critical state, then stream(T, m) can only have a symbol at the ith position of the form $(\langle N' \rangle, \mathsf{false})$ for some tag N', by property (P4). In addition, property (P2) implies that $i \neq pos(m)$ in this case. Hence, we know $\delta(q'', (\langle N \rangle, \mathsf{false}))$ iff $i \neq pos(m)$ holds as required. If q'' is not a critical state, then it is select-indifferent by property (P3), and hence both disjuncts of the transition in qtrans(p) are satisfied (recall the definition of qtrans(p)). If w is a descendant of v in $\mathcal{F}$, then it is select-indifferent by property (P4), and hence again the appropriate disjunct is satisfied. If w is neither an ancestor nor a descendant of v, then let w' be their least common ancestor, and consider the successors of w' in $\mathcal{F}$. By property (P5), at most one of these can reach a critical state, and this must be the ancestor of v. Hence, in particular, w is select-indifferent, and the conclusion follows.

Conversely, suppose that $T \models q(n)$. We show that there is an accepting run of qtrans(p) on the string stream(T) at position $pos(n)$. By induction we know that there exists a node m and an accepting run of trans(p) on the string stream(T, m) at position $pos(n)$. It is easy to construct from this an accepting run of qtrans(p) starting at position $pos(n)$.

This completes the proof of Claim 7.6.

7.3.6. *Enforcing the Correct Form of Strings.* So far, we assumed that strings given as input of the 2WA(S)As are of the form stream(T) or stream(T, m), where T is a tree satisfying a non-recursive DTD D. We now show how to code such a DTD D in terms of a 2WAA. We say that a 2WAA $\mathcal{A}_D$ *weakly enforces* D if (a) for every tree T satisfying D, stream(T) is accepted by $\mathcal{A}_D$, and (b) if a string s is accepted by A, then $s =$ stream(T) for some tree T satisfying D.

CLAIM 7.7. *Given a nonrecursive DTD D, there is a 2WAA $\mathcal{A}_D$ that weakly enforces D, which can be found in PTIME.*

PROOF OF CLAIM. Indeed, let *Ele* be the element set of D. For every element $N \in Ele$, we can write an ordinary finite state machine that enforces the input string such that, whenever we see a symbol $\langle N \rangle$, then sometime afterwards there is a symbol $\langle /N \rangle$, and in between those there are no additional occurrences of $\langle N \rangle$. Let $\mathcal{A}_N$ be this automaton and s_N its initial state. Then, the 2WAA $\mathcal{A}_D$ is the disjoint union of $\mathcal{A}_N$ over all $N \in Ele$ together with the initial formula $\theta_0 = \bigwedge_{N \in Ele} s_N$. That is, we verify each of these properties in parallel. The fact that D is nonrecursive guarantees that this is correct.

We remark that we can obtain a similar 2WASA $\mathcal{A}_D$ for strings of the form stream(T, m). The difference is that we also need to enforce that there is unique symbol ($\langle N \rangle$, true) in the accepted strings.

We also need to enforce that only streams satisfying the DTD are accepted. For this, we observe that we can encode the *nonrecursive* DTD D in terms of a qualifier q_D in $\mathcal{X}(\downarrow, \uparrow, \downarrow^*, \uparrow^*, \leftarrow, \rightarrow, \leftarrow^*, \rightarrow^*, \cup, [\ ], \neg)$ such that for any tree T, $root(T)$ satisfies q_D iff T satisfies D. As stated in Claim 7.6, q_D can be encoded in terms of a 2WAA trans(q_D).

This completes the proof of Claim 7.7.

7.3.7. *Completion of the Proof.* Given D and p, we define the 2WASA $\mathcal{A}_{D,p}$ as the conjunction of trans(p), $\mathcal{A}_D$ and trans(q_D). Here conjunction refers to taking the disjoint union of the automata and conjoining the initial formulas.

From Claims 7.6, 7.7, and the construction of q_D, it is clear that testing nonemptiness of $\mathcal{A}_{D,p}$ is equivalent to testing the satisfiability of (D, p). Since testing the nonemptiness of two-way alternating automata is known to be in PSPACE, we have the desired upper bound.

Recall that the PSPACE algorithm consists of building an equivalent one-way nondeterministic automaton. Although this requires exponential time (see, e.g., kupferman et al. [2001]), the one-way automata can be explored in PSPACE, by building it "on-the-fly" (see, e.g., Appendix A.1 of Martens and Neven [2005]; note that all of our automata are "loop-free" in the sense of Martens and Neven [2005]). This is sufficient for testing nonemptiness since this is essentially checking for the reachability of an accepting state. □

We next show that for any XPath fragment $\mathcal{X}(\cup, \dots)$ that contains '$\cup$' but does not allow '$\downarrow^*$, $\uparrow^*$', there is a quadratic time reduction from SAT($\mathcal{X}(\cup, \dots)$) under *arbitrary* DTDs to SAT($\mathcal{X}(\cup, \dots)$) under *nonrecursive* DTDs.

LEMMA 7.8. *Let $\mathcal{X}(\cup, \dots)$ be any XPath fragment that contains union but does not allow the descendant and ancestor axes. Then, there exists a function f : $\mathcal{X}(\cup, \dots) \rightarrow \mathcal{X}(\cup, \dots)$, computable in $O(|p|^2)$ time, and moreover, for each query $p \in \mathcal{X}(\cup, \dots)$, there exists a function NR_p from normalized DTDs to nonrecursive normalized DTDs, computable in $O(|D||p|)$ time, such that for any DTD D, (p, D) is satisfiable if and only if ($f(p)$, $NR_p(D)$) is satisfiable.*

PROOF. We first introduce a notion of depth: For any query $p \in \mathcal{X}(\cup, \dots)$ and any tree T, we identify the depth of the deepest node in T involved in the evaluation of p. More specifically, we define the *depth of* p, denoted by depth(p), inductively on the structure of p as follows: (a) $\mathsf{depth}(\varepsilon) = \mathsf{depth}(\rightarrow) = \mathsf{depth}(\leftarrow) = \mathsf{depth}(\rightarrow^*) = \mathsf{depth}(\leftarrow^*) = \mathsf{depth}(\uparrow) = 0$; (b) $\mathsf{depth}(A) = \mathsf{depth}(\downarrow) = 1$ for any label A in p; (c) $\mathsf{depth}(p_1/p_2) = \mathsf{depth}(p_1) + \mathsf{depth}(p_2)$;

(d) $\mathsf{depth}(p_1 \cup p_2) = \max\{\mathsf{depth}(p_1), \mathsf{depth}(p_2)\}$; (e) $\mathsf{depth}(p_1[q]) = \mathsf{depth}(p_1) + \mathsf{depth}([q])$; (f) $\mathsf{depth}([q]) = \mathsf{depth}(q)$; (g) $\mathsf{depth}(q_1 \wedge q_2) = \mathsf{depth}(q_1 \vee q_2) = \max\{\mathsf{depth}(q_1), \mathsf{depth}(q_2)\}$; (h) $\mathsf{depth}(\neg q_1) = \mathsf{depth}(q_1)$; (i) $\mathsf{depth}(\mathsf{lab}() = A) = 0$; (j) $\mathsf{depth}(p/@a = c) = \mathsf{depth}(p)$; and (k) $\mathsf{depth}(p/@a \text{ op } p'/@b) = \max\{\mathsf{depth}(p), \mathsf{depth}(p')\}$.

We now define the function $f : \mathcal{X}(\cup, \dots) \to \mathcal{X}(\cup, \dots)$ that maps each query p to an expression $f(p) \in \mathcal{X}(\cup, \dots)$. Let $n = \mathsf{depth}(p)$, and for each label A occurring in p, we introduce n new labels A^i, $i \in [1, n]$. We define $f(p)$ by rewriting p as follows: (1) replacing each label A occurring in p by the expression $(A^1 \cup \cdots \cup A^n)$; and (2) replacing each label test $\mathsf{lab}() = A$ in p by $(\mathsf{lab}() = A^1) \vee \cdots \vee (\mathsf{lab}() = A^n)$. The function f is clearly computable in $O(|p|^2)$ time.

Let T be an XML document and let $\mathsf{depth}_T()$ be the function that assigns to each node v of T its the depth in T, that is, the length of the path in T from v up to the root of T. We denote this by $\mathsf{depth}_T(v)$. We define the *layered version* of T as the XML document T_ℓ obtained from T by replacing the label of each node v in T by the label $\mathsf{lab}(n)^{\mathsf{depth}_T(v)}$. It can be easily verified that $T \models p$ if and only if $T_\ell \models f(p)$. In fact, if we define $T_\ell^{\leq n}$ to be the subtree of T_ℓ obtained by removing all nodes from T_ℓ below depth n, then we also have that $T \models p$ if and only if $T_\ell^{\leq n} \models f(p)$.

Let $D = (Ele, Att, P, R, r)$. We now define the mapping NR_p that maps D to a non-recursive DTD $NR_p(D)$. More specifically, let $NR_p(D) = (Ele', Att', P', R', r')$, where $Ele' = \{A^i \mid A \in Ele, i \in [1, n]\} \cup \{r'\}$; $Att' = Att$; $r' = r^0$; $R'(A^i) = R(A)$; and for $i \in [1, n-1]$, P' consists of the productions (a) $P'(A^i) = B_1^{(i+1)} + \cdots + B_k^{(i+1)}$ if $P(A) = B_1 + \cdots + B_k$, (b) $P'(A^i) = B_1^{(i+1)}, \dots, B_k^{(i+1)}$ if $P(A) = B_1, \dots, B_k$, and (c) $P'(A^i) = (B^{(i+1)})^*$ if $P(A) = B^*$. A special case is the production $P'(r^0)$. If, for example, $P(r) = B_1, \dots, B_k$, then $P'(r^0) = B_1^1, \dots, B_k^1$; similarly if $P(r) = B_1 + \ldots + B_k$ or $P(r) = B^*$. Finally, we extend P' by adding the productions $P'(A^n) = \varepsilon$ for all labels $A \in Ele$. Clearly, $NR_p(D)$ is a nonrecursive DTD and the function NR_p is computable in $O(|D||p|)$ time.

It is easy to verify that if $T \models D$, then $T_\ell^{\leq n} \models NR_p(D)$. Moreover, if $T \models (p, D)$, then $T_\ell^{\leq n} \models (f(p), NR_p(D))$. Conversely, if a tree $T' \models (f(p), NR_p(D))$ we can obtain a tree $T \models (p, D)$ by (1) replacing each label A^i in T' by A; and (2) extending T' to a tree that conforms to D, by applying productions in D to the leaf nodes in T'. □

7.4. SATISFIABILITY WITH SIBLING AXES AND DATA VALUES. Similarly, undecidability results can be obtained for $\mathsf{SAT}(\mathcal{X}(\uparrow, \leftarrow, \rightarrow, \rightarrow^*, \cup, [\,], =, \neg))$ under restricted DTDs and in the absence of DTDs.

THEOREM 7.9. $\mathsf{SAT}(\mathcal{X}(\uparrow, \leftarrow, \rightarrow, \rightarrow^*, \cup, [\,], =, \neg))$ *is undecidable either* (1) *under non-recursive, fixed and disjunction-free DTDs; or* (2) *in the absence of DTDs.*

PROOF. We prove these by reduction from the halting problem for two-register machines (2RM). Recall from the proof of Theorem 5.4 the description of 2RM machines M and the statement of the halting problem.

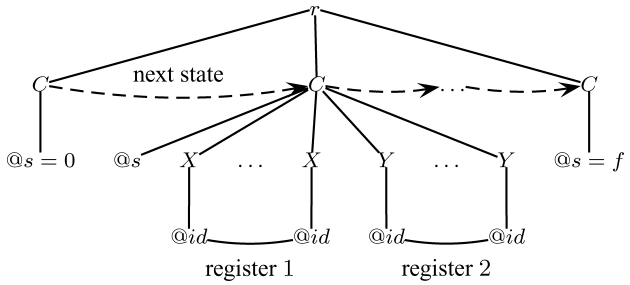

FIG. 13. Illustration of the DTD used in the undecidability proof of $\mathsf{SAT}(\mathcal{X}(\uparrow, \leftarrow, \rightarrow, \rightarrow^*, \cup, [\,], =, \neg))$.

1. Under non-recursive, fixed and disjunction-free DTDs. We first define the fixed DTD $D = (Ele, Att, P, R, r)$ as follows:

$Ele = \{r,\ C,\ X,\ Y\}$.
P: $\quad r \to C^*, \qquad C \to X^*, Y^*$.
$Att = \{@s,\ @id\}, \quad R(C) = \{@s\}, R(X) = R(Y) = \{@id\}$.

As depicted in Figure 13, an XML tree of the DTD D consists of an unbounded number of C children, which encodes the execution of a 2RM M starting from the leftmost C element. Each C element encodes an ID of M: it has an s attribute indicating the state of the ID, and a list of X children followed by a list of Y children on the right. The number of X (respectively, Y) children represents the contents of $\mathsf{register}_1$ (respectively, $\mathsf{register}_2$). To count the number of X children with the same parent, an $X.id$ attribute is defined for X elements, which is to serve as a *local key* for these X elements. Similarly, for the Y children.

To encode M we use the following $\mathcal{X}(\uparrow, \leftarrow, \rightarrow, \rightarrow^*, \cup, [\,], =, \neg)$-qualifiers at the root r.

(1) *Initial ID*. We code the initial ID $(0, 0, 0)$ of M by using the leftmost C child of r.

$$Q_{start} = C[(\varepsilon/@s = 0\ \wedge\ \neg(\leftarrow \vee X \vee Y))].$$

(2) *Halting state*. The final ID $(f, 0, 0)$ of M is expressed as

$$Q_{halting} = Q_{start}/{\rightarrow^*}[(\varepsilon/@s = f\ \wedge\ \neg(\rightarrow \vee X \vee Y))].$$

This asserts that there exists an ID (i.e., a C child of r) reachable from the initial ID such that it is $(f, 0, 0)$, and moreover, no more computation is conducted (or ID is reached) after M enters $(f, 0, 0)$.

(3) *Local key.* We enforce $X.id$ to be a local key for the set of X children:

$$Q_{xKey} = \neg C[X/@id = \rightarrow/{\rightarrow^*}[\mathsf{lab}() = X]/@id],$$

This asserts that under any C element, the $X.id$ of any X element is different from the $X.id$ of any of its X siblings; in other words, all the X children of C have distinct $X.id$ values. Similarly, we define Q_{yKey} for Y elements.

(4) *Transition*. For each $i \in [0, l]$, we code the ith instruction I_i in terms of a qualifier Q_i, based on the type of I_i.

(Case 1: *Addition*). If I_i is an addition transition (i, rg, j), where $\mathsf{rg} = \mathsf{register}_1$, then Q_i is defined to ensure that *for any* C element c_1 with state $c_1.s = i$, (i) its

right sibling c_2 has state $c_2.s = j$ (state change); (ii) c_2 has one more X child than c_1 ($\mathsf{register}_1$ is incremented by 1); and moreover, (iii) c_2 has the same number of Y children as c_1 ($\mathsf{register}_2$ remains unchanged). These are expressed as follows:

$$\begin{aligned}
Q_i &= \neg C[\varepsilon/@s = i \ \wedge (\rightarrow/@s \neq j \ \vee \ Q_a^X \ \vee \ Q^Y)] \\
Q_a^X &= (X[\neg(\varepsilon/@id = \uparrow/\rightarrow/X/\rightarrow^*[\mathsf{lab}() = X]/@id)]) \quad \vee \\
&\qquad (\rightarrow/X/\rightarrow^*[\mathsf{lab}() = X \wedge \rightarrow[\mathsf{lab}() = X] \wedge \neg(\varepsilon/@id = \uparrow/\leftarrow/X/\rightarrow^*[\mathsf{lab}() = X]/@id)]) \\
Q^Y &= (Y[\neg(\varepsilon/@id = \uparrow/\rightarrow/Y/\rightarrow^*/@id)]) <_{\text{sib}} \vee(\rightarrow/Y[\neg(\varepsilon/@id = \uparrow/\leftarrow/Y/\rightarrow^*/@id)]).
\end{aligned}$$

Here, Q_i, Q_a^X and Q^Y assert the conditions (i–iii) above. Similarly, Q_i, Q_a^X and Q^Y can be defined for $\mathsf{rg} = \mathsf{register}_2$.

(Case 2: *Subtraction*). If I_i is a subtraction (i, rg, j, k), where $\mathsf{rg} = \mathsf{register}_1$, then Q_i is defined to ensure that *for any* C element c_1 with state $c_1.s = i$, (i) its right sibling c_2 has state $c_2.s = j$ if c_1 has no X child (i.e., $\mathsf{register}_1$ is 0), and furthermore, c_2 has no X child and it has the same number of Y children as c_1; and (ii) if c_1 has an X child (i.e., $\mathsf{register}_1 \neq 0$), then c_2 has state $c_2.s = k$, and moreover, c_2 has one less X child than c_1 ($\mathsf{register}_1$ is decremented by 1), while the number of Y children of c_2 is the same as that of c_1 ($\mathsf{register}_2$ remains unchanged). These are expressed as follows:

$$\begin{aligned}
Q_i &= \neg C\big[\varepsilon/@s = i \ \wedge (Q_s^0 \ \vee \ Q_s^X)\big] \\
Q_s^0 &= (\neg X \ \wedge \ (\rightarrow/@s \neq j \ \vee \ \rightarrow[X] \ \vee \ Q^Y)) \\
Q_s^X &= (X \ \wedge \ (\rightarrow/@s \neq k \ \vee \ X[\rightarrow^*[\mathsf{lab}() = X \wedge \rightarrow[\mathsf{lab}() = X] \quad \wedge \\
&\qquad \neg(\varepsilon/@id = \uparrow/\rightarrow/X/\rightarrow^*[\mathsf{lab}() = X]/@id)]) \quad \vee \\
&\qquad\qquad (\rightarrow/X[\neg(\varepsilon/@id = \uparrow/\leftarrow/X/\rightarrow^*[\mathsf{lab}() = X]/@id)])
\end{aligned}$$

Here, Q^Y is the same as defined in Case 1, and Q_i, Q_s^0 and Q_s^X assert the conditions (i – ii) above. Similarly, Q_i, Q_s^0 and Q_s^X can be defined for $\mathsf{rg} = \mathsf{register}_2$.

Putting these together, we define the query Q in $\mathcal{X}(\uparrow, \leftarrow, \rightarrow, \rightarrow^*, \cup, [\,], =, \neg)$ to be

$$Q = \varepsilon\left[Q_{start} \ \wedge \ Q_{halting} \ \wedge \ Q_{xKey} \ \wedge \ Q_{yKey} \ \wedge \ \bigwedge_{i\in[0,l]} Q_i\right].$$

One can verify that (Q, D) is satisfiable iff the 2RM M halts, that is, $(0, 0, 0) \Rightarrow_M (f, 0, 0)$.

2. In the absence of DTDs. We encode the DTD D given in part (1) using qualifiers. First, the production $P(r)$ can be easily expressed by a qualifier saying that there should be a C child of the root, followed only by C labeled right siblings:

$$Q_r = C[\neg(\leftarrow) \wedge \neg(\rightarrow^*[\mathsf{lab}() \neq C])].$$

Second, for the production $P(C)$ we encode a mild variation $P(C) = S, X^*, Y^*$ since then we know that C has at least one child to which we can go down (we do not have $\downarrow$ in the XPath fragment). It can be easily verified that the undecidability proof of (1) goes through after this minor modification. We now encode (the modified) $P(C)$ with the following qualifier, asserting that there should be a single leftmost

S child below C, followed by only X and Y labeled right siblings in the right order (Y after X):

$$Q_C = \neg C[S[\leftarrow \vee \rightarrow/\rightarrow^*[\mathsf{lab}() \neq X \wedge \mathsf{lab}() \neq Y]] \wedge \neg S] \wedge \neg C[S/\rightarrow/\rightarrow^*[\mathsf{lab}() = Y]/\rightarrow/\rightarrow^*[\mathsf{lab}() = X]].$$

One can verify that for Q in the proof of (1), $\varepsilon[Q_r \wedge Q_C]/Q$ is satisfiable if and only if (Q, D) is satisfiable if and only if the 2RM M halts. □

7.5. CONTAINMENT ANALYSIS. Obviously, Proposition 3.2 still holds in the presence of the sibling axes. From this and the lower bounds established in this section, it follows:

COROLLARY 7.10. *For the containment problem,*

(1) $\mathsf{CNT}(\mathcal{X}(\rightarrow, [\,], \neg))$ *and* $\mathsf{CNT}(\mathcal{X}(\leftarrow, [\,], \neg))$ *are PSPACE-hard (a) under non-recursive and no-star DTDs, and (b) in the absence of DTDs;*

(2) $\mathsf{CNT}(\mathcal{X}(\uparrow, \rightarrow, [\,], \neg))$ *is EXPTIME-hard under fixed, disjunction-free and non-recursive DTDs;*

(3) $\mathsf{CNT}(\mathcal{X}(\uparrow, \leftarrow, \rightarrow, \rightarrow^*, \cup, [\,], =, \neg))$ *is undecidable (a) under non-recursive, disjunction-free and fixed DTDs, and (b) in the absence of DTDs.*

These are among the first lower bound results for containment analysis of XPath with sibling axes. Indeed, the only other result that we are aware of is the EXPTIME-hardness given by Marx [2004] for $\mathsf{CNT}(\mathcal{X}(\downarrow, \downarrow^*, \cup, [\,], \neg))$. Corollary 7.10 strengthens that result by showing that $\mathsf{CNT}(\mathcal{X}(\uparrow, \rightarrow, [\,], \neg))$ is already EXPTIME-hard under restricted DTDs.

Furthermore, Proposition 3.2 also remains intact in the presence of the sibling axes. Indeed, the function $\mathsf{inverse}$ in the proof of Proposition 3.2 can be extended by including $\mathsf{inverse}(\rightarrow) = \leftarrow$, $\mathsf{inverse}(\rightarrow^*) = \leftarrow^*$, $\mathsf{inverse}(\leftarrow) = \rightarrow$ and $\mathsf{inverse}(\leftarrow^*) = \rightarrow^*$, and the rest of that proof can be carried over. In this way, additional complexity results for the containment problem in the presence of sibling axes can be obtained.

8. *Conclusion*

We have studied the satisfiability problem for a variety of XPath fragments in the presence of DTDs, in the absence of DTDs, and under restricted DTDs. The main complexity results are summarized in Table I, annotated with their corresponding theorems.

Under arbitrary (any) DTDs, the table shows that the complexity of $\mathsf{SAT}(\mathcal{X})$ ranges from PTIME to NP-complete when $\mathcal{X}$ is a positive XPath fragment. When negation is added, the complexity ranges from PSPACE-complete to undecidable, depending on different combinations of the negation operator, the recursive axes and data-value joins.

Under nonrecursive (nonrec) DTDs, the satisfiability problem becomes much simpler for XPath fragments with recursive axes; however, the absence of DTD recursion does not help the satisfiability analysis of XPath fragments without recursive axes. The absence of disjunction (+-free) in DTDs simplifies the satisfiability analysis of positive XPath fragments, but does not help fragments with negation. Fixing the DTD has little impact on the worst-case analysis, while the absence

TABLE I. THE MAIN RESULTS: THE COMPLEXITY OF $\mathsf{SAT}(\mathcal{X})$ FOR VARIOUS FRAGMENTS $\mathcal{X}$ UNDER DIFFERENT DTDS

$\downarrow$	$\downarrow^*$	$\uparrow$	$\uparrow^*$	$\cup$	[ ]	$=$	$\neg$	any DTDs	nonrec. DTDs	fixed DTDs	'+'-free DTDs	DTD-free
+	+			+				PTIME (Th 4.1)	PTIME (Th 4.1)	PTIME (Th 4.1)	PTIME (Th 4.1)	PTIME (Th 3.1, 4.1)
				+	+			NP-complete (Th 4.4)	NP-complete (Th 6.3, 4.4)	NP-complete (Th 6.6, 4.4)	PTIME (Th 6.8)	PTIME (Th 6.11)
+					+			NP-complete (Th 4.4)	NP-complete (Th 6.3, 4.4)	NP-complete (Th 6.6, 4.4)	PTIME (Th 6.8)	PTIME (Th 6.11)
+		+						NP-complete (Th 4.4)	NP-complete (Th 6.3, 4.4)	NP-complete (Th 6.6, 4.4)	PTIME (Th 6.8)	PTIME (Th 6.11)
+	+			+	+			NP-complete (Th 4.4)	NP-complete (Th 6.3, 4.4)	NP-complete (Th 6.6,4.4)	PTIME (Th 6.8)	PTIME (Th 6.11)
+					+	+		NP-complete (Th 4.4)	NP-complete (Th 6.3, 4.4)	NP-complete (Th 6.6, 4.4)	NP-complete (Th 6.9, 4.4)	PTIME (Th 6.11)
				+	+	+		NP-complete (Th 4.4)	NP-complete (Th 6.3, 4.4)	NP-complete (Th 6.6, 4.4)	NP-complete (Th 6.9, 4.4)	NP-complete (Th 6.14, 4.4)
+		+		+	+			NP-complete (Th 4.4)	NP-complete (Th 6.3, 4.4)	NP-complete (Th 6.9, 4.4)	NP-complete (Th 6.9, 4.4)	NP-complete (Th 6.14, 4.4)
+	+	+	+	+	+	+		NP-complete (Th 4.4)	NP-complete (Th 6.3, 4.4)	NP-complete (Th 6.6, 4.4)	NP-complete (Th 6.9, 4.4)	NP-complete (3.1, 6.14, 4.4)
+					+		+	PSPACE-com -plete (Th 5.2)	PSPACE-com -plete (6.2, 6.3)	PSPACE-com -plete (6.7, 5.2)	PSPACE-com -plete (6.10, 5.2)	PSPACE-com -plete (6.15,5.2)
+		+		+	+		+	PSPACE-com -plete (Th 5.2)	PSPACE-com -plete (6.2, 6.3)	PSPACE-com -plete (6.7, 5.2)	PSPACE-com -plete (6.10, 5.2)	PSPACE-com -plete (6.15,5.2)
+	+				+		+	EXPTIME-com -plete (Th 5.3)	PSPACE-com -plete (6.2, 6.3)	EXPTIME-com -plete (6.7, 5.3)	EXPTIME-com -plete (6.10, 5.3)	EXPTIME-com -plete (6.15,5.3)
+	+	+	+	+	+		+	EXPTIME-com -plete (Th 5.3)	PSPACE-com -plete (6.2, 6.3)	EXPTIME-com -plete (6.7, 5.3)	EXPTIME-com -plete (6.10, 5.3)	EXPTIME-com -plete (6.15,5.3)
		+			+	+	+	EXPTIME-hard (Th 5.6)	EXPTIME-hard (Cor 6.3)	EXPTIME-hard (Th 6.7)	EXPTIME-hard (Cor 6.10)	EXPTIME-hard (Cor 6.15)
+				+	+	+	+	NEXPTIME (Th 5.5)	NEXPTIME (Th 5.5)	NEXPTIME (Th 5.5)	NEXPTIME (Th 5.5)	NEXPTIME (Th 3.1, 5.5)
+	+	+	+	+	+	+	+	undecidable (Th 5.4)	?	undecidable (Th 6.7)	?	?

of DTDs (DTD-free) diminishes the complexity for positive fragments but not for those fragments with negation.

We have also investigated the satisfiability problem for XPath fragments with sibling axes (those results are not included in Table I). Our main conclusion here is that the presence of sibling axes does not complicate the satisfiability analysis.

There is still much to be done for the analysis of XPath fragments with negation and data values. An open question is the complexity of $\mathsf{SAT}(\mathcal{X}(\downarrow, \uparrow, \downarrow^*, \uparrow^*, \cup, [\ ], =, \neg))$ that is, the largest fragment with recursive and upward axes, negation and data-value joins, in the absence of DTDs, DTD disjunctions, or DTD recursion. We only know that the problem is undecidable in the absence of DTDs or DTD disjunctions if we allow position() or equality on node identity. Another issue is the decidability of $\mathsf{SAT}(\mathcal{X}(\downarrow, \downarrow^*, \cup, [\ ], =, \neg))$, that is, the largest downward fragment. We only know that the problem is decidable either under non-recursive DTDs, or in absence of the recursive axis ($\downarrow^*$). It would also be interesting to see if the NEXPTIME bound for nonrecursive DTDs remains valid when queries have access to a document ordering.

ACKNOWLEDGMENTS. We thank Kousha Etessami and Kedar Namjoshi for helpful discussions, and Peter Buneman and Leonid Libkin for their comments. We also thank the referees for suggestions on improving the article.

REFERENCES

AFANASIEV, L., BLACKBURN, P., DIMITRIOU, I., GAIFFE, B., GORIS, E., MARX, M., AND DE RIJKE, M. 2005. PDL for ordered trees. *J. Non-Classical Logics 15*, 115–135.

AMER-YAHIA, S., CHO, S., LAKSHMANAN, L., AND SRIVASTAVA, D. 2002. Tree pattern query minimization. *VLDB J. 11*, 4, 315–331.

BENEDIKT, M., FAN, W., AND GEERTS, F. 2005. XPath satisfiability in the presence of DTDs. In *Proceedings of the 24th ACM Symposium on Principles of Database Systems (PODS)*. ACM, New York, 25–36.

BENEDIKT, M., FAN, W., AND KUPER, G. M. 2005. Structural properties of XPath fragments. *Theoret. Comput. Sci. 336*, 1, 3–31.

BÖRGER, E., GRÄDEL, E., AND GUREVICH, Y. 1997. *The Classical Decision Problem*. Springer-Verlag, New York.

BRAY, T., PAOLI, J., AND SPERBERG-MCQUEEN, C. M. 1998. Extensible markup language (XML) 1.0. W3C Recommendation. `http://www.w3.org/TR/REC-xml`.

CALVANESE, D., DE GIACOMO, G., LENZERINI, M., AND VARDI, M. Y. 2001. View-based query answering and query containment over semistructured data. In *Revised Papers of the 8th International Workshop on Database Programming Languages (DBPL 2001)*, G. Ghelli and G. Grahne, Eds. Lecture Notes in Computer Science, vol. 2397. Springer-Verlag. New York, 40–61.

CHAMBERLIN, D., BOAG, S., FERNÁNDEZ, M. F., FLORESCU, D., ROBIE, J., AND SIMÉON, J. 2001. XQuery 1.0: An XML query language. W3C Working Draft.

CHAN, C., FELBER, P., GAROFALAKIS, M., AND RASTOGI, R. 2002. Efficient filtering of XML documents with XPath expressions. In *Proceedings of the 18th IEEE International Conference on Data Engineering (ICDE)*. IEEE Computer Society Press, Los Alamitos, CA, 235–244.

CHLEBUS, B. S. 1986. Domino-tiling games. *J. Comput. Syst. Sci. (JCSS) 32*, 3, 374–392.

CLARK, J., AND DEROSE, S. 1999. *XML Path Language (XPath)*. W3C Recommendation.

DEUTSCH, A., AND TANNEN, V. 2005. XML queries and constraints, containment and reformulation. *Theoret. Comput. Sci. 336*, 1, 57–87.

FAN, W., CHAN, C., AND GAROFALAKIS, M. 2004. Secure XML querying with security views. In *Proceedings of the 23th ACM International Conference on Management of Data (SIGMOD)*. ACM, New York, 587–598.

FAN, W., AND LIBKIN, L. 2002. On XML integrity constraints in the presence of DTDs. *J. ACM 49*, 3, 368–406.

GEERTS, F., AND FAN, W. 2005. Satisfiability of XPath queries with sibling axes. In *Proceedings of the 10th International Symposium on Database Programming Languages (DBPL)*. Lecture Notes in Computer Science, vol. 3774. Springer-Verlag, New York, 122–137.

GOTTLOB, G., KOCH, C., AND PICHLER, R. 2005. Efficient algorithms for processing XPath queries. *ACM Trans. Datab. Syst. 30*, 2, 444–491.

HIDDERS, J. 2004. Satisfiability of XPath expressions. In *Proceedings of the 9th International Workshop on Database Programming Languages (DBPL)*. Lecture Notes in Computer Science, vol. 2921. Springer-Verlag, New York, 21–36.

HOPCROFT, J. E., AND ULLMAN, J. D. 2000. *Introduction to Automata Theory, Languages and Computation (2nd Edition)*. Addison Wesley. Reading, MA.

JAGADISH, H., LAKSHMANAN, L., SRIVASTAVA, D., AND THOMPSON, K. 2002. TAX: A tree algebra for XML. In *Proceedings of the 8th International Workshop on Database Programming Languages (DBPL)*. Lecture Notes in Computer Science, vol. 2397. Springer-Verlag, New York, 149–164.

KUPERFERMAN, O., PITERMAN, N., AND VARDI, M. Y. 2001. Extended temporal logic revisited. Lecture Notes in Computer Science, vol. 2154. Springer-Verlag, New York, 519–535.

LAKSHMANAN, L., RAMESH, G., WANG, H., AND ZHAO, Z. 2004. On testing satisfiability of tree pattern queries. In *Proceedings of the 30th International Conference on Very Large Data Bases (VLDB)*. Morgan Kaufmann, San Francisco, CA, 120–131.

MARTENS, W., AND NEVEN, F. 2004. Frontiers of tractability for typechecking simple XML transformations. In *Proceedings of the 23rd ACM Symposium on Principles of Database Systems (PODS)*. ACM, New York, 23–34.

MARTENS, W., AND NEVEN, F. 2005. On the complexity of typechecking top-down xml transformations. *Theoretical Computer Science 336*, 153–180.

MARX, M. 2004. XPath with conditional axis relations. In *Proceedings of the 9th International Conference on Extending Database Technology (EDBT)*. Lecture Notes in Computer Science, vol. 2992. Springer-Verlag, New York, 477–494.

MARX, M., AND DE RIJKE, M. 2005. Semantic characterizations of navigational XPath. *SIGMOD Record 34*, 2, 41–46.

MIKLAU, G., AND SUCIU, D. 2004. Containment and equivalence for a fragment of XPath. *J. ACM 51*, 1, 2–45.

MILO, T., SUCIU, D., AND VIANU, V. 2003. Typechecking for XML transformers. *J. Comput. Syst. Sci. 66*, 1, 66–97.

MURATA, M. 2001. Extended path expressions for XML. In *Proceedings of the 20th Symposium on Principles of Database Systems (PODS)*. ACM, New York, 126–137.

NEVEN, F., AND SCHWENTICK, T. 2000. Expressive and efficient pattern languages for tree-structured data. In *Proceedings of the 19th ACM Symposium on Principles of Database Systems (PODS)*. ACM, New York, 145–156.

NEVEN, F., AND SCHWENTICK, T. 2006. On the complexity of XPath containment in the presence of disjunction, DTDs, and variables. *Log. Meth. Comput. Sci. 2*, 3, 1–30.

OLTEANU, D., MEUSS, H., FURCHE, T., AND BRY, F. 2002. XPath: Looking forward. In *Proceedings of Workshop on XML-Based Data Management and Multimedia Engineering (XMLDM)*. Lecture Notes in Computer Science, vol. 2490. Springer-Verlag, New York, 109–127.

PAPADIMITRIOU, C. H. 1994. *Computational Complexity*. Addison-Wesley, Reading, MA.

PAPARIZOS, S., WU, Y., LAKSHMANAN, L., AND JAGADISH, H. 2004. Tree logical classes for efficient evaluation of XQuery. In *Proceedings of the 23rd ACM International Conference on Management of Data (SIGMOD)*. ACM, New York, 71–82.

STOCKMEYER, L. 1974. The complexity of decision problems in automata theory and logic. Tech. rep. MIT, Cambridge, MA.

SUR, G., HAMMER, J., AND SIMÉON, J. 2004. An XQuery-based language for processing updates in XML. In *Proceedings of Workshop on Programming Language Technologies for XML (Plan-X)*. 40–53.

THATCHER, J., AND WRIGHT, J. 1968. Generalized finite automata with an application to a decision problem of second-order logic. *Math. Syst. Theory 2*, 57–82.

VARDI, M., AND WOLPER, P. 1986. Automata theoretic techniques for modal logics of programs. *J. Comput. Syst. Sci. 32*, 183–221.

W3C. 2006. *XML Path Language (XPath) 2.0*. W3C Recommendation.

WOOD, P. 2001. Minimising simple XPath expressions. In *Proceedings of the 4th International Workshop on the Web and Databases (WebDB)*. 13–18.

WOOD, P. 2002. Containment for XPath fragments under DTD constraints. In *Proceedings of the 9th International Conference (ICDT)*. Lecture Notes in Computer Science, vol. 2572. Springer-Verlag, New York, 300–314.